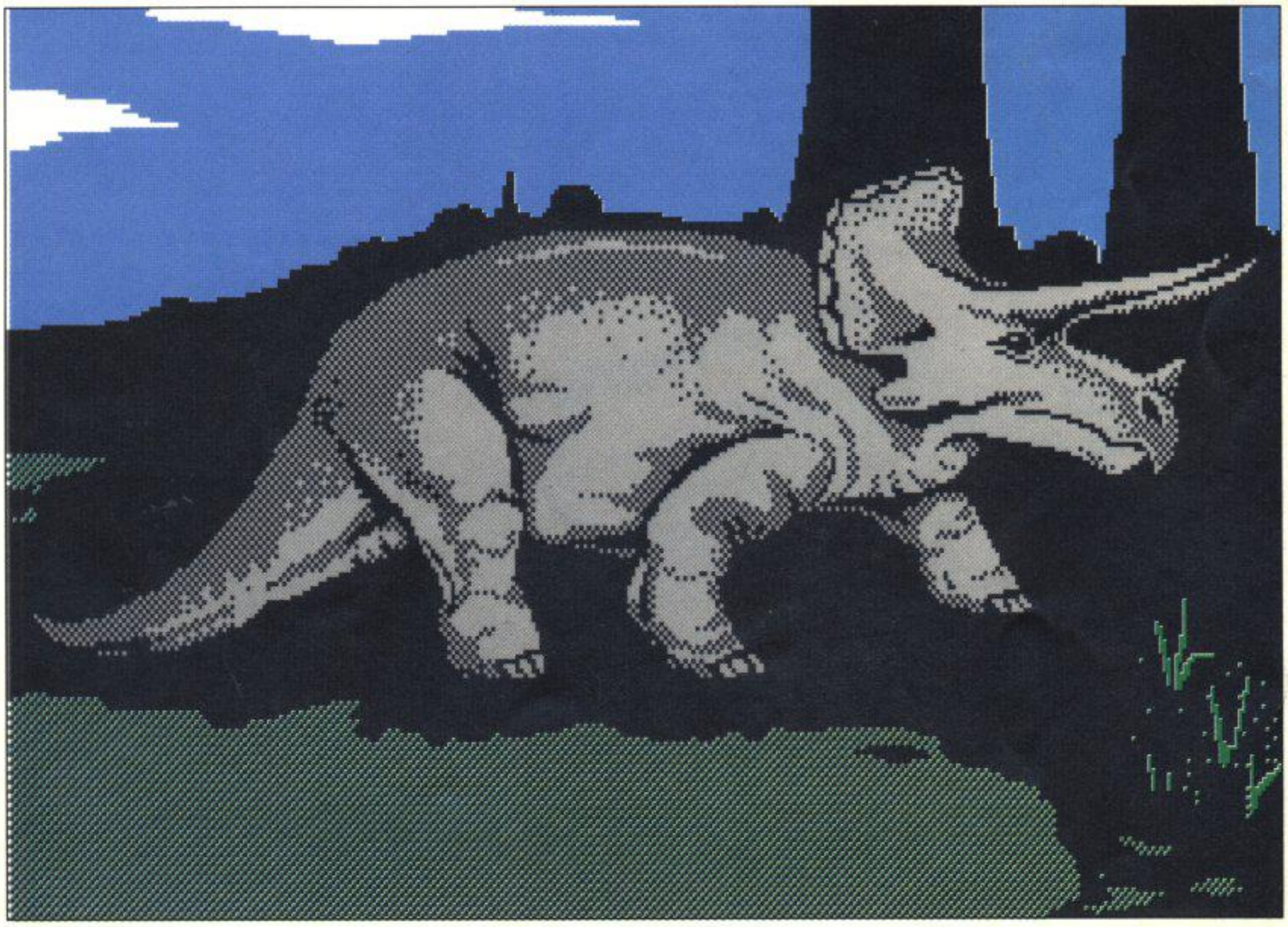
SEPTEMBER 1988: VOLUME 9, ISSUE 1

Canada $4.25
USA $3.50

Transactor

- C128 scrolling directory utility
- Multitasking on the C128
- Exploring CP/M's SUBMIT program
- The ultimate machine language input routine
- Fast ML sprite rotation
- An *I Ching* Hexagram generator
- C64 hexadecimal file editor
- Notes on C programming and RAM expansion units
- Programming in GEOS
- **Plus** Reviews of the Lt. Kernal hard drive and Cinemaware's Warp Speed cartridge; tips and programs in *bits*; new product announcements and more.



Trilemma by Jo-Anne Park

THE MAGAZINE FOR COMMODORE PROGRAMMERS

0 29074 12121 9

NORTHWEST MUSIC CENTER INC.

539 N. Wolf Rd. — Wheeling, IL 60090
Hours— (Voice) Phone (312) 520-2540
Mon.-Thurs. 12:30-5:00, Sat. 12:00-4:00
(24 Hour Order Recorder)

We want to be your Commodore shop!

Do you feel like no one cares about you and your orphaned computer! NWM cares!! We still stock and sell most major B series and 8000 series programs.

Parts available for some 8000, 9000 and B series models!

Hacker's Corner

One of a Kind • Surplus • Monthly Special • Closeouts

Limited quantities to stock on hand

64 k ieee to parallel buffer	$199.00	8023p and MPP 1361 ribbons	$5.50
4023p 100cps rehab	$99.00	9090 7.5 meg rehab	$495.00
64k ram exp 8032	$110.00	4023p ribbons	$6.00
Smith Corona DM-200	$179.95	Everex 2400 external modem	$245.00
6400, 8300p, Diablo 630 ribbons	$4.95		

NWM's INVENTORY CONTROL SYSTEM*

- loads program modules in less than **8 seconds** (superbase 2) to main menus in **3 seconds** or less
- on screen **pop-up calculator** in transaction modules
- most data centered function use the **calculator keypad**
- versatile report features allow for **3 ways** to print the same report. User selects the fastest method
- built in sophisticated export program allows for **complete packing** of the database
- type ahead feature allowed
- you can display reports on screen
- access to superbase menu for user developed applications

B Version 1 8050 . . . . . $39.95
B Version 2 8050 . . . . . $39.95
C-128 Version 1 1571 . . . . . $24.95
B-128 Version 1&2 8050 . . . . . $44.95

*Requires use of superbase

B-128 $145 U.S.

NEW 128K USER INSTALLABLE MEMORY EXPANSION!
INTRODUCTORY PRICE OF ONLY $125.

SOFTWARE FOR THE B-128!!!

Superbase	$19.95	C.A.B.S. Accounting	
Superscript	$19.95	General Ledger	$ 9.95
Superoffice Integrated		Accounts Receivable	$ 9.95
Superbase & Superscript	$49.95	Accounts Payable	$ 9.95
Calc Result	$89.95	Order Entry	$ 9.95
Word Result	$89.95	Payroll	$ 9.95
Super Disk Doc	24.95	Buy all 5 for only $24.95	
The Power of: Calc Result (Book)	$14.95	Superbase: The Book	$14.95
		Applied Calc Result (Book)	$14.95

Commodore's Superpet 9000

only $199.00

while supplies last

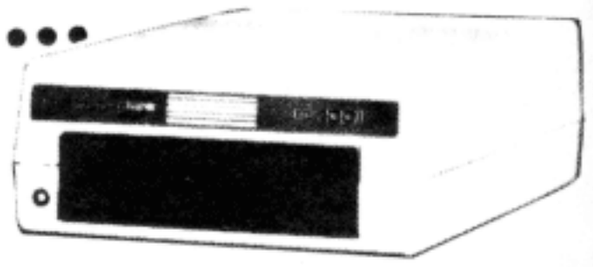
With Five Interpretive Languages:
Cobol
Pascal
Basic
Fortran
Apl

Runs 8032 software.
Great for schools and students

64K Memory Expansion for 8032 only $110 upgrades your 8032 to an 8096.

COMMODORE 8000-9000 SOFTWARE & MISC.

9000 Superpet	$199.00	Superscript 8032	$79
64K exp for 8032	$110	Superbase 8096	$79
OZZ Database	$25	Legal Time Acc	$25
BPI General Ledger	$25	Dow Jones Program	$25
BPI Accts Payable	$25	Info Designs 8032	
BPI Job Cost	$25	Accounting System	$50
BPI Accts Receivable	$25	Superoffice 8096	$149
BPI Inventory	$25	Calc Result 8032	$89



SFD 1001 1 Megabyte
PRICED AT **$149.95** (US)
$125 with purchase of Superpet

SFD-1001 is the drive that you should consider when you need large amounts of data storage. It holds over 1 megabyte of data on its single floppy drive. Fast IEEE access for your C-64 or C-128. (C-64 and C-128 need an IEEE interface.) Why settle for slower drives with less storage capacity. This drive stores substantially more programs and data. Think how much money you can save on disk purchases. In fact, it stores almost 7 times more information than your standard drive. Bulletin board owners love them. And what an introductory price! At $169.95 these drives will sell fast, so don't wait. This drive has the identical format of a CBM 8250 drive, one of Commodores most durable floppy drives.

MODEL	SFD-1001	Sector/Cylinder	—
DRIVES	1	Sector/Track	23-29
HEADS/DRIVE	2	Bytes/Sector	256
STORAGE CAPACITY (Per Unit)		Free Blocks	4133
Formatted	1.06 Mb	TRANSFER RATES (Bytes/Sec)	
MAXIMUM (Each Drive)		Internal	40 Kb
Sequential File	1.05 Mb	IEEE-488 Bus	1.2 Kb
Relative File	1.04 Mb	ACCESS TIME (Milli-seconds)	
Disk System		Track-to-track	*
Buffer RAM (Bytes)	4K	Average track	**
DISK FORMATS (Each Drive)		Average Latency	100
Cylinders (Tracks)	(77)	Speed (RPM)	300

ORDER NOW WHILE STOCK LASTS!

Send or call your orders to: **Northwest Music Center, Inc. 539 N. Wolf Rd., Wheeling IL 60090. 312-520-2540** For prepaid orders add $25.50 for Superpet, $10.45 SFD 1001, $11.45 B-128, $10.45 4023p, $16.45 9090 and $5.45 64K memory expansion. For software add $3.50 for first and $2.00 for each additional book or program. Canadian shipping charges are double U.S. For C.O.D. orders add $2.20 per box shipped. All orders must be paid in U.S. funds. Include phone numbers with area codes. Do not use P.O. Box, only UPS shippable addresses. A 2 week hold will be imposed on all orders placed with a personal or business check. C.O.D. orders shipped in U.S. only and cash on delivery, no checks. 30 day warranty on all products from NWM, Inc. No manufacturer warranty. NWM reserves the right to limit quantities to stock on hand and adjust prices without notice!

All prices quoted in US dollars.

"What's a G-Link?"

Glad you asked.

A G-Link is an interface that lets you expand your 64 to an incredibly powerful system by letting you connect all of Commodore's IEEE disk drives and peripherals. With the G-Link, you can use devices from the fast, reliable 4040 dual drive to the high-storage SFD-1001 single drive in just the same way as you use your 1541. You can switch between the serial devices you're using now and the G-Link supported IEEE devices with the flick of a switch. And the G-Link is transparent, meaning it won't interfere with the operation of most software: as far as your computer is concerned, you're just using a super-fast 1541 drive!

If you're ready to upgrade your system, or you have some IEEE equipment you want to connect (which includes printers, plotters and all kinds of scientific instruments), consider the inexpensive G-Link. Order from the card in the centrefold.

**** COMMODORE ****

parts & service
DAVE TAYLOR ENTERPRISES
4400 N. Big Spring #3 * Midland, TX 79705
915-686-0535

FLAT RATE REPAIRS

C64 repair	$45.00
1541	$50.00
C128 repair	$65.00
1571 repair	$75.00
SX64 repair	$75.00

MISCELLANEOUS PARTS

C64 power supply	$19.95
C128 power supply (repairable)	$84.95
C64 power supply (repairable)	$34.95
SX64 transformer	$29.95
C64 Diagnostics	$150.00
C128 Diagnostics	$150.00
1541 Diagnostics	$165.00
Diagnostician (C-64/1541/1571)	$6.95

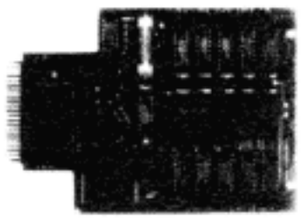
**** CALL FOR COMPLETE PARTS PRICE LIST ****

All prices F.O.B. Midland Tx. Prices subject to change without notice. Texas residents add 7.5% sales tax. All sales and repairs carry a **(30) day warranty.** except IC's which have been pre-tested. repairs do not include external power supplies or return shipping. please add 3% for *VISA/MC* charges.

915-686-0535

UNLEASH THE DATA ACQUISITION AND CONTROL POWER OF YOUR COMMODORE C64 OR C128.
***We have the answers* to all your control needs.**

NEW! 80-LINE SIMPLIFIED DIGITAL I/O BOARD

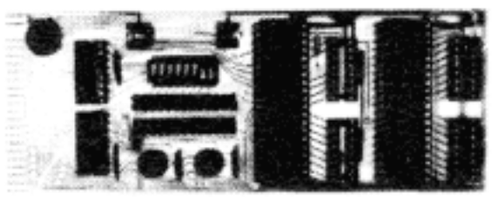


Create your own autostart dedicated controller without relying on disk drive.

- Socket for standard ROM cartridge.
- 40 separate buffered digital output lines can each directly switch 50 volts at 500 mA.
- 40 separate digital input lines. (TTL).
- I/O lines controlled through simple memory mapped ports each accessed via a single statement in Basic. No interface could be easier to use. A total of ten 8-bit ports.
- Included M.L. driver program optionally called as a subroutine for fast convenient access to individual I/O lines from Basic.
- Plugs into computer's expansion port. For both C64 & C128. I/O connections are through a pair of 50-pin professional type strip headers.
- Order Model SS100 Plus. **Only $119!** Shipping paid USA. Includes extensive documentation and program disk. Each additional board $109.

We take pride in our interface board documentation and software support, which is available separately for examination. Credit against first order.
SS100 Plus, $20. 64IF22 & ADC0816, $30.

OUR ORIGINAL ULTIMATE INTERFACE



- Universally applicable dual 6522 Versatile Interface Adapter (VIA) board.
- Industrial control and monitoring. Great for laboratory data acquisition and instrumentation applications.
- Intelligently control almost any device.
- Perform automated testing.
- Easy to program yet extremely powerful.
- Easily interfaced to high-perfomance A/D and D/A converters.
- Four 8-bit fully bidirectional I/O ports & eight handshake lines. Four 16-bit timer/counters. Full IRQ interrupt capability. Expandable to four boards.

Order Model 64IF22. $169 postpaid USA. Includes extensive documentation and programs on disk. Each additional board $149. Quantity pricing available. For both C64 and C128.

A/D CONVERSION MODULE

Fast. 16-channel. 8-bit. Requires above. Leaves all VIA ports available. For both C64 and C128. Order Model 64IF/ADC0816. Only $69.

SERIOUS ABOUT PROGRAMMING?

SYMBOL MASTER MULTI-PASS SYMBOLIC DISASSEMBLER. Learn to program like the experts! Adapt existing programs to your needs! Disassembles any 6502/6510/undoc/65C02/8502 machine code program into beautiful source. Outputs source code files to disk fully compatible with your MAE, PAL, CBM, Develop-64, LADS, Merlin or Panther assembler, ready for re-assembly and editing. Includes both C64 & C128 native mode versions. 100% machine code and extremely fast. 63-page manual. The original and best is now even better with Version 2.1! Advanced and sophisticated features far too numerous to detail here. $49.95 postpaid USA.

C64 SOURCE CODE. Most complete available reconstructed, extensively commented and cross-referenced assembly language source code for Basic and Kernal ROMs, all 16K. In book form, 242 pages. $29.95 postpaid USA.

PTD-6510 SYMBOLIC DEBUGGER for C64. An extremely powerful tool with capabilities far beyond a machine-language monitor. 100-page manual. Essential for assembly-language programmers. $49.95 postpaid USA.

MAE64 version 5.0. Fully professional 6502/65C02 macro editor/assembler. 80-page manual. $29.95 postpaid USA.

NEW ADDRESS!

SCHNEDLER SYSTEMS
Dept. 91, 25 Eastwood Road, P.O. Box 5964
Asheville, North Carolina 28813 Telephone 1-704-274-4646

NEW ADDRESS!

All prices in U.S. dollars.

Transactor

The Magazine for Commodore Programmers

Volume 9, Issue 1

Publisher
Richard Evers

Editors
Malcolm O'Brien
Nick Sullivan
Chris Zamara

Editorial Assistant
Moya Drummond

Customer Service
Renanne Turner

Accounting
Donna Evers

Contributing Writers
Ian Adam
Jack Bedard
Bill Brier
Jim Butterfield
Don Currie
Jim Frost
Miklos Garamszeghy
Eric Giguere
David Godshall
Thomas Gurley
Adam Herst
D. J. Morriss
Gary Kiziak
Bob Kodadek
Francis Kostella
Keath Milligan
Mike Mohilo
Noel Nyman
Adrian Pepper
Steve Punter
Tony Romer
Herb Rose
Audrys Vilkas

Cover Artist
Jo-Anne Park

ScrollDir 128 — 15

by Miklos Garamszeghy

The ultimate directory utility - scroll up and down through your file names, load programs, display text, and scratch files without typing

Multitasking on the Commodore 128 — 20

by Mike Mohilo

Run up to four programs simultaneously, or switch between tasks instantly - even BASIC can run in the background!

Exploring SUBMIT — 24

by Adam Herst

Adam's look at one of the most useful tools in CP/M Plus goes far deeper than the docs

A Machine Language Input Routine — 28

by Garry Kiziak

The bullet-proof, all-purpose, high-performance, configurable, easy-to-use input routine

Sprite Rotation — 36

by Jim Frost

A super-fast ML implementation of *Transactor*'s "sprite rotate" - a boon for video game programmers

Structured DATA and Seeding RND — 42

by Audrys Vilkas

Something *completely* different: *I Ching*, yin and yang, Hexagrams, Ancient Chinese farmers... and random numbers

C64 Hex File Editor — 46

by Bob Kodadek

Edit disk files in memory, machine language monitor-style

On the C Side... — 54

by Adrian Pepper

Insights into C programming on the 64 and 128

Programming in GEOS — 56

by Francis G. Kostella

How to get your machine language applications to run under GEOS

Departments and Columns

Letters 7

Bits 11

G-Link on newer computers
Self-Save
Find Joy
Hook, Line and Singer
Easy 128 Key Fix
POKE Poser Figured Out
Data Checker 64
Late Night TV
Re-Booting GEOS 128
Never-never land 128D

News BRK 77

C128 developer's package
Mystic Jim's stuff
1988 Commodore Computerfest
Computer Save
Micro Detective professional debugger
Super 81 Utilities for the C64
C128 complete bookkeeping system
Romjet custom cartridge update
Superboot for C128
Satellite tracking program
Anatomy of the 4040 disk drive
CP/M Starter Set from PD Solutions

Reviews

Lt. Kernal Hard Drive 67

by Bill Brier

Super power for the 64 with this fast, feature-laden hard drive system

The 1351 Mouse and GEOS 1.3 72

GEOS was never this easy

Warp Speed 74

Cinemaware's multi-purpose cartridge brings you far beyond mere impulse power. Engage!

About the cover: We're getting just a little bit tired of hearing 8-bit computers like the Commodore 64 and 128 referred to as 'dinosaurs', so for this issue's cover we asked Toronto artist Jo-Anne Park to remind the 16/32-bit crowd what dinosaurs really look like. Even at a casual glance you can see there's really very little resemblance to any microcomputer, even an Atari.

Jo-Anne specializes in Commodore 64 and Amiga art. She did the cover for an upcoming issue of *Transactor for the Amiga*, and we liked her work so much we asked her to do a *Transactor* cover - using the C64 - as well. The picture was done using Doodle, so it's hi-res rather than multi-colour; as C64 graphic artists are aware, creating good colour graphics in hi-res mode is quite a challenge. Through the ingenuity of creative people, the 8-bit machines will continue to be viable for a long time to come

Transactors's
phone number is:
(416) 764-5273
Line open Mondays, Wednesdays
and Fridays ONLY

FAX: (416) 764-9262

TOLL-FREE ORDER LINE
1-800-248-2719 Extension 911
(for orders only; have your VISA or Mastercard number ready; available in the U.S. only)

Transactor is published bimonthly by Transactor Publishing Inc., 85-10 West Wilmot Street, Richmond Hill, Ontario, L4B 1K7. ISSN# 0838-0163. Canadian Second Class Mail Registration No. 7690, Gateway-Mississauga, Ont. US Second Class mail permit pending at Buffalo, NY. USPS Postmasters: send address changes to: Transactor, PO Box 338, Station C, Buffalo, NY, 14209.

Transactor Publishing Inc. is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names are registered trademarks of Commodore Inc.

Subscriptions:

Canada $19 Cdn.
USA $15 US
All others $21 US
Air Mail (Overseas only) $40 US

Send all subscriptions to: Transactor Publishing Inc., Subscriptions Department, 85-10 West Wilmot Street, Richmond Hill, Ontario, Canada, L4B 1K7, (416) 764-5273. For best results, use the postage paid card at the centre of the magazine.

Quantity Orders: In **Canada**: Ingram Software Ltd., 141 Adesso Drive, Concord, Ontario, L4K 2W7, (416) 738-1700. In the **USA**: IPD (International Periodical Distributors), 11760-B Sorrento Valley Road, San Diego, California, 92121, (619) 481-5928; ask for Dave Buescher. Quantity orders/enquiries are also welcome from computer/software distributors in the **UK, Europe and Scandinavia**. Please contact: T.G. Hamilton (W/S) Ltd., Tel: 021-742-5359; Fax: 021-742-2190; or contact Transactor (UK) direct at Unit 2, Langdale Grove, Bingham, Notts. NG13 8SR, telephone 0949-39380. In **Australia**, contact Transactor (Australia) Pty. Limited, 35 Calder Cr., Holder, ACT 2611, Australia. Phone 61 62 883584.

Editorial contributions are welcome. Only original, previously unpublished material will be considered. Program listings and articles, including BITS submissions, of more than a few lines, should be provided on disk. Preferred format is 1541-format with ASCII text files. Manuscripts should be typewritten, double-spaced, with special characters or formats clearly marked. Photos should be glossy black and white prints. Illustrations should be on white paper with black ink only. Hi-res graphics files on disk are preferred to hardcopy illustrations when possible.

All material accepted becomes the property of Transactor Publishing Inc., except by special arrangement. All material is copyright by Transactor Publishing Inc. Reproduction in any form without permission is in violation of applicable laws. Write to the Richmond Hill address for a writer's guide.

The opinions expressed in contributed articles are not necessarily those of Transactor Publishing Inc. Although accuracy is a major objective, Transactor Publishing Inc. cannot assume liability for errors in articles or programs. Programs listed in *Transactor*, and/or appearing on *Transactor* disks, are copyright by Transactor Publishing Inc. and may not be duplicated or distributed without permission.

Production
In-house with Amiga 2000 and Professional Page

Final output by Vellum Print & Graphic Services, Inc., Toronto

Printing
Printed in Canada by Maclean Hunter Printing

Using "VERIFIZER"

Transactor's foolproof program entry method

VERIFIZER should be run before typing in any long program from the pages of *Transactor*. It will let you check your work line by line as you enter the program and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line; you can then check this code against the corresponding one in the printed program listing.

There are three versions of VERIFIZER here: one each for the PET/CBM, VIC/C64, and C128 computers. Enter the applicable program and RUN it. If you get a data or checksum error, re-check the program and keep trying until all goes well. You should SAVE the program since you'll want to use it every time you enter a program from *Transactor*. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 634 to enable the PET/CBM version (off: SYS 637)
SYS 828 to enable the C64/VIC version (off: SYS 831)
SYS 3072,1 to enable the C128 version (off: SYS 3072,0)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute, changing the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough: transposing two sets of four characters will produce the same report code, but this will rarely happen. (VERIFIZER could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking the program manually). VERIFIZER ignores spaces so you may add or omit spaces from the listed program at will (providing you don't split up keywords!) Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI   10 rem* data loader for "verifizer 4.0" *
LI   20 cs=0
HC   30 for i=634 to 754: read a: poke i,a
DH   40 cs=cs+a: next i
GK   50 :
OG   60 if cs<>15580 then print"***** data error *****": end
JO   70 rem sys 634
AF   80 end
IN   100 :
ON  1000 data  76, 138,   2, 120, 173, 163,   2, 133, 144
IB  1010 data 173, 164,   2, 133, 145,  88,  96, 120, 165
CK  1020 data 145, 201,   2, 240,  16, 141, 164,   2, 165
EB  1030 data 144, 141, 163,   2, 169, 165, 133, 144, 169
HE  1040 data   2, 133, 145,  88,  96,  85, 228, 165, 217
OI  1050 data 201,  13, 208,  62, 165, 167, 208,  58, 173
JB  1060 data 254,   1, 133, 251, 162,   0, 134, 253, 189
PA  1070 data   0,   2, 168, 201,  32, 240,  15, 230, 253
HE  1080 data 165, 253,  41,   3, 133, 254,  32, 236,   2
EL  1090 data 198, 254,  16, 249, 232, 152, 208, 229, 165
LA  1100 data 251,  41,  15,  24, 105, 193, 141,   0, 128
KI  1110 data 165, 251,  74,  74,  74,  74,  24, 105, 193
EB  1120 data 141,   1, 128, 108, 163,   2, 152,  24, 101
DM  1130 data 251, 133, 251,  96
```

VIC/C64 VERIFIZER

```
KE  10 rem* data loader for "verifizer" *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print"***** data error *****": end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP  1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC  1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN  1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG  1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM  1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA  1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG  1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK  1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN  1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH  1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC  1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP  1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH  1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH  1140 data 101,  89, 133,  89,  96
```

***NEW* C128 VERIFIZER (40 or 80 column mode)**

```
KL  100 rem save"0:c128 vfz.ldr",8
OI  110 rem c-128 verifizer
MO  120 rem bugs fixed:  1) works in 80 column mode.
DG  130 rem              2) sys 3072,0 now works.
KK  140 rem
GH  150 rem by joel m. rubin
HG  160 rem * data loader for "verifizer c128"
IF  170 rem * commodore c128 version
DG  180 rem * works in 40 or 80 column mode!!!
EB  190 ch=0
GC  200 for j=3072 to 3220: read x: poke j,x: ch=ch+x: next
NK  210 if ch<>18602 then print "checksum error": stop
BL  220 print "sys 3072,1 to enable
DP  230 print "sys 3072,0 to disable
AP  240 end
BA  250 data 170, 208,  11, 165, 253, 141,   2,   3
MM  260 data 165, 254, 141,   3,   3,  96, 173,   3
AA  270 data   3, 201,  12, 240,  17, 133, 254, 173
FM  280 data   2,   3, 133, 253, 169,  39, 141,   2
IF  290 data   3, 169,  12, 141,   3,   3,  96, 169
FA  300 data   0, 141,   0, 255, 165,  22, 133, 250
LC  310 data 162,   0, 160,   0, 189,   0,   2, 201
AJ  320 data  48, 144,   7, 201,  58, 176,   3, 232
EC  330 data 208, 242, 189,   0,   2, 240,  22, 201
PI  340 data  32, 240,  15, 133, 252, 200, 152,  41
FF  350 data   3, 133, 251,  32, 141,  12, 198, 251
DE  360 data  16, 249, 232, 208, 229,  56,  32, 240
CB  370 data 255, 169,  19,  32, 210, 255, 169,  18
OK  380 data  32, 210, 255, 165, 250,  41,  15,  24
ON  390 data 105, 193,  32, 210, 255, 165, 250,  74
OI  400 data  74,  74,  74,  24, 105, 193,  32, 210
OD  410 data 255, 169, 146,  32, 210, 255,  24,  32
PA  420 data 240, 255, 108, 253,   0, 165, 252,  24
BO  430 data 101, 250, 133, 250,  96
```

The Standard Transactor Program Generator

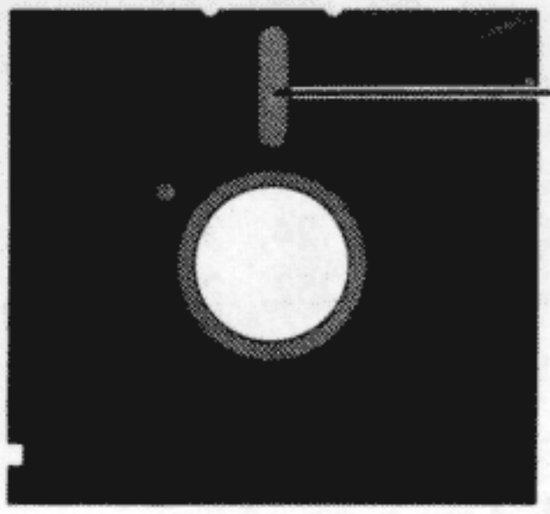
If you type in programs from the magazine, you might be able to save yourself some work with the program listed on this page. Since many programs are printed in the form of a BASIC "program generator" which creates a machine language (or BASIC) program on disk, we have created a "standard generator" program that contains code common to all program generators. Just type this in once, and save all that typing for every other program generator you enter!

Once the program is typed in (check the Verifizer codes as usual when entering it), save it on a disk for future use. Whenever you type in a program generator, the listing will refer to the standard generator. Load the standard generator *first*, then type the lines from the listing as shown. The resulting program will include the generator code and be ready to run.

When you run the new generator, it will create a program on disk (the one described in the related article). The generator program is just an easy way for you to put a machine language program on disk, using the standard BASIC editor at your disposal. After the file has been created, the generator is no longer needed. The standard generator, however, should be kept handy for future program generators.

The standard generator listed here will appear in every issue from now on (when necessary) as a standard *Transactor* utility like Verifizer.

```
MG  100 rem  transactor standard program generator
EE  110 n$="filename": rem  name of program
LK  120 nd=000: sa=00000: ch=00000
KO  130 for i=1 to nd: read x
EC  140 ch=ch-x: next
FB  150 if ch then print "data error": stop
DE  160 print "data ok, now creating file."
CM  170 restore
CH  180 open 1,8,1,"0:"+n$
HM  190 hi=int(sa/256): lo=sa-256*hi
NA  200 print#1,chr$(lo)chr$(hi);
KD  210 for i=1 to nd: read x
HE  220 print#1,chr$(x);: next
JL  230 close 1
MP  240 print"prg file '";n$;"' created..."
MH  250 print"this generator no longer needed."
IH  260 :
```



Start Address

Evolution in the Eight-Bit World

First, a brief note: no, you haven't missed an issue, this one really is about two months late. That's pretty late for a bi-monthly magazine, and it certainly doesn't do much to instill confidence in readers and advertisers (both current and potential). We thought we'd let you know what's going on, and why you can believe us when we say we're back on track now.

Production-wise, we now have a schedule that guarantees that a magazine gets produced in 56 days, barring unforseeable catastrophes of biblical proportions. This issue remained unprinted for so long due to financial difficulties within the company (read: not enough money) that have since been cleared up with an influx of capital and business know-how. Our spreadsheet shows good news ahead, so the reliable production schedule is backed by a financially stable company. Newsstand circulation has just increased again as we appear on the shelves in Waldenbooks in the U.S., so the 8-bit Transactor is still growing even as the 16-bit computers increase their presence in the market.

Enough talk of the real world: we take you now to the originally scheduled editorial for this issue.

* * * * *

With this issue we welcome a new member to our editorial staff, which brings us up to a three-man team. Malcolm O'Brien has been with us a few months now, and has had much to do with the creation of this magazine. We think you'll like the flavour that Malcolm's touch brings to the magazine, as there will be more focus on real-world and "power-user" applications; as you can see already, GEOS will no longer be a stranger to these pages. The following editorial, written by Malcolm himself, will complete the introduction.

Evolution means changes. Lots of changes. Probably more of them for me than for you. I've gone from a nine to five, two subway stop, merchandising, strictly IBM job to an all hours, long haul, 90 per cent 8-bit, 10 per cent Amiga, editing job. I'm starting to learn the ropes around the office and on CompuServe. It's a strange environment but it fits me well. Hmm...

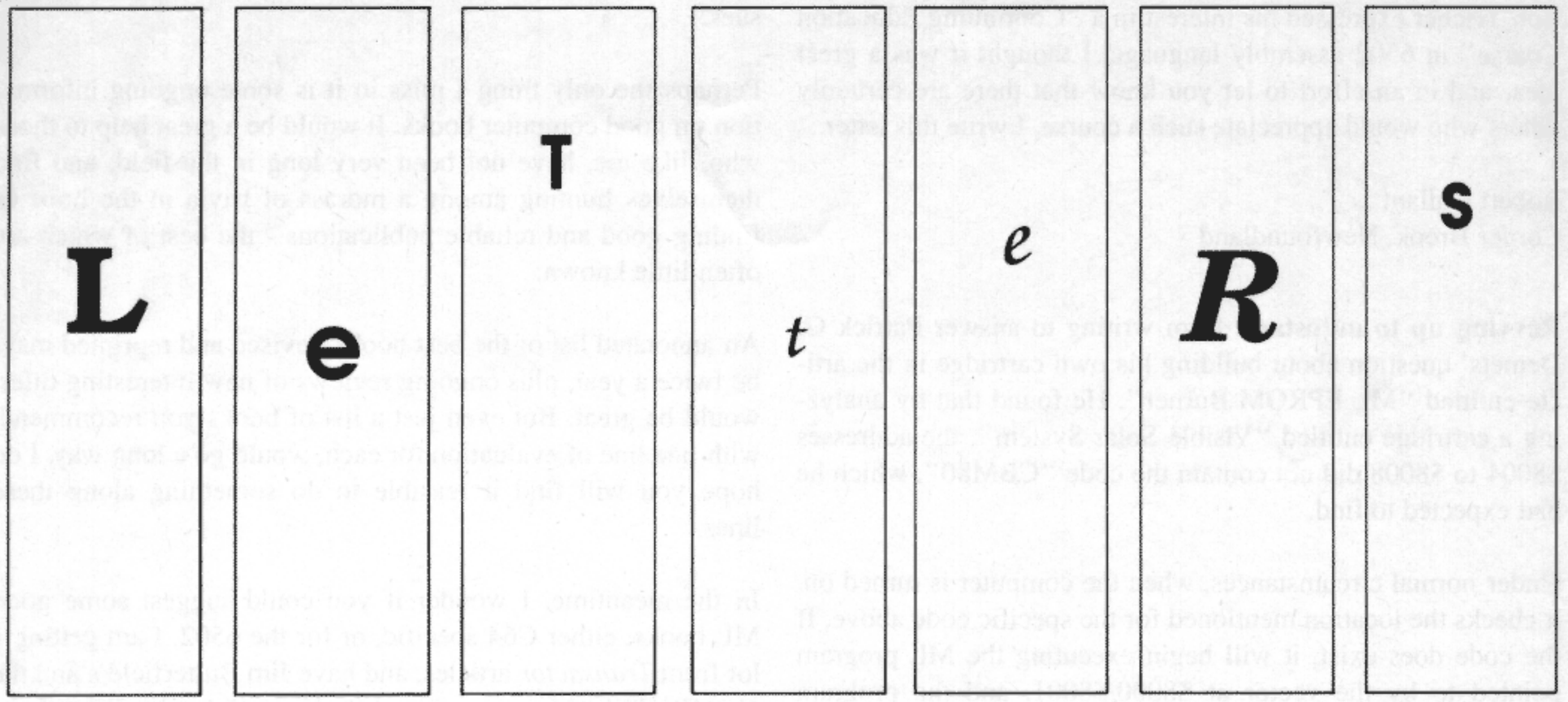
The changes *you* will experience will be of a different nature. For one thing, we're going to be doing GEOS coverage. We've had letters requesting it and some submissions. Expect to see articles on GEOS programming, starting with this issue. For the many people who've requested ML subroutines, we have *two* in this issue - one for the tricky job of sprite rotation, and another that is perhaps the ultimate in configurable input routines.

We'll also be featuring articles with a "pushing the limits" theme. These will be concerned with doing things that are often considered to be beyond the capabilities of the 8-bit machines. For a sample of what I mean, take a look at the Lt. Kernal article in this issue.

You've already noticed the inclusion of C coverage. Response to this has been uniformly favourable and we'll continue doing it. This is a recognition of the fact that many C64 and C128 users are also using other languages on other computers at work, at school and at home. C is the main one but there are others as well. Coverage for other languages such as COMAL will probably be appearing in *Transactor*s of the future.

These are significant changes but they reflect the ongoing evolution of the user base. Haven't we all been reminded for years that "there's nothing as constant as change"? It's true - even though they told us in programming school that constants weren't supposed to change.

Malcolm O'Brien



It's about time: This is a reply to a letter published in *Transactor* ("Clock Setting", Letters, Volume 8, Issue 5), where reader David Kuhn briefly describes his computerized light and automatic sprinkling system controller and queries whether the C128 can read an external real time clock to reset its internal clocks following a power failure.

An excellent product, which should do exactly what the reader (and many others) requires, is the Model CCSZ Cartridge from Jason-Ranheim, 1805 Industrial Drive, Auburn, California, USA 95603. Their phone numbers are (800) 421-7731 and (916) 823-3284. Their price is $49.95 (US), plus shipping.

The CCSZ not only includes a battery-backed Clock / Calendar, but 8K of battery-backed RAM and a modified operating system in ROM to support the features. The CCSZ can automatically download and run a program when power resumes following an outage, and even maintains a power-off/power-on log in RAM. Moreover, the cartridge, which works in both the C64 and C128 (in the C128 mode), will automatically recognize which computer it is being used with.

Now for the commercial message: The CCSZ from Jason-Ranheim is fully compatible with the control interface boards which we (Schnedler Systems) manufacture for the C64 and C128, and we believe many readers will be interested in both as a compatible system. Our Model SS100 Plus 80-line Simplified Digital I/O Board is particularly attractive in this regard because it includes a standard 44-pin cartridge socket for receiving cartridges such as the Jason-Ranheim CCSZ, as well as standard EPROM cartridges. Thus the SS100 Plus may be viewed as a digital data acquisition and control interface combined with a single-slot expansion motherboard. The price currently is still only $119.00 (US), including the manual and program disk. Shipping to US addresses is included in that price. For shipping to Canada add $10.00, and add $20.00 to other countries.

Steven C. Schnedler
Schnedler Systems
25 Eastwood Rd.
P.O. Box 5964
Asheville, NC 28813
(704) 274-4646

Time backed-up: The problem that David Kuhn expressed in the Letters column of Volume 8, Issue 5, can be overcome by relating to a previous article in *Transactor*. In Volume 6, Issue 6, Jean Des Rosiers, the author of "Home Control on a VIC" interfaced various hardware projects for a home security/controller run by a VIC-20. Amongst these projects was a battery back-up in case of power failure. This involved alkaline batteries added into the power supply circuit. The schematic diagrams are in Figures 5 through 7, inclusive, on page 70 of that issue.

I made a similar project and used the schematics from this project to have the battery back-up. I hope to upgrade this to NiCads with current "steering" diodes.

I feel that this sort of device is what's needed with David's C128. He will still have to obtain the schematic of the 128's power supply to know what has to be added.

Daryl Leopold
Vancouver, British Columbia

Another M/L aficionado: In a recent letter to your magazine, Bob Tischer expressed his interest in a "Continuing Education Course" in 6502 assembly language. I thought it was a great idea, and in an effort to let you know that there are certainly others who would appreciate such a course, I write this letter.

Robert Gallant
Corner Brook, Newfoundland

Revving up to autostart: I am writing to answer Patrick G. Demets' question about building his own cartridge in the article entitled "ML EPROM Burner". He found that by analyzing a cartridge entitled "Visible Solar System", the addresses $8004 to $8008 did not contain the code "CBM80", which he had expected to find.

Under normal circumstances, when the computer is turned on, it checks the location mentioned for the specific code above. If the code does exist, it will begin executing the ML program pointed to by the vector at $8000/$8001, and the program pointed to by the $8002/$8003 vector will be the warmstart procedure. If the code is not present in locations $8004 to $8008, however, control is given to the program pointed to by the vector at $a000/$a001, and the warmstart is pointed to by $a002 to $a003. Locations $a004 to $a008 need not contain the code CBM80. An autostart cartridge addressed for $a000 to $bfff will replace BASIC.

The above procedure is the common method of cartridge design. Its mode of operation is quite simple. On power-up, the 6510 microprocessor jumps to the ML program pointed to by $fffc/$fffd. This program has many tasks, including testing for an autostart cartridge.

Mr. Demets mentions that all of the JMP's and JSR's are targetted at locations beginning with $e. The 64 has a third address range for cartridges. This is $e000 to $ffff. Evidently, Mr. Demets' cartridge occupied this range along with $8000 to $9fff (perhaps even $a000 to $bfff). The autostart program pointed to by locations $fffc to $fffd is located at $fce2. However, a cartridge with the address range $e000 to $ffff will replace the computers own memory at this location. Therefore, there will be a new startup vector at $fffc/$fffd (not to mention a new NMI at $fffa/$fffb and a new IRQ/BRK vector at $fffe/$ffff). The new vector may point to any other memory location, but wherever it does point, that is the new autostart program. If the 64 autostart program is replaced, the code at locations $8004 to $8008 is irrelevant (unless, of course, the new autostart program calls for it).

Bernard Epsilon Wolfe
Oakville, Ontario

Book List?: First of all, I would like to express my appreciation for your magazine. I find *Transactor* to be consistently excellent for quality, technical level, usefulness of material, friendliness, and in many other ways. (I have been reading it for about one year so far, and just ordered all available back issues.)

Perhaps the only thing I miss in it is some ongoing information on good computer books. It would be a great help to those who, like me, have not been very long in the field, and find themselves hunting among a morass of trivia in the hope of finding good and reliable publications - the best of which are often little known.

An annotated list of the best books, revised and reprinted maybe twice a year, plus ongoing reviews of new interesting titles, would be great. But even just a list of books you recommend, with one line of evaluation for each, would go a long way. I do hope you will find it feasible to do something along these lines.

In the meantime, I wonder if you could suggest some good ML books, either C64 specific, or for the 6502. I am getting a lot from *Transactor* articles, and have Jim Butterfield's and the *COMPUTE!* "Mapping..." and "...Kernal" books. What I am looking for is, say, the equivalent of the Neufeld and Immers' books, for ML programming. I hope you can help. Thank you.

James G. Vargiu
Atlanta, Georgia

Sounds like you already have a pretty fair collection, James. Jim Butterfield's book is an excellent introduction to machine language programming, and ***Mapping the Commdore 64*** *is also very useful. If you're looking for some hard-core reference material, you might also be interested in* ***The Complete Commodore Inner Space Anthology****, which is published by a very reputable Canadian company (the one that publishes* ***Transactor****, strangely enough). The* ***CCISA*** *has been around for a long time now, long enough that it doesn't cover the C128, but it's still a gold mine of concentrated information on the other 8-bit Commodore machines. As for an annotated list - well, how about it readers? What are your favourites, and why?*

All Together Now: I would like to suggest a new and useful way to use the 7,000,000+ Commodore 64 and 128 computers. It is by using them for a parallel processing project. First I will describe Project #1.

It has long been suspected that Pierre Fermat was right when he wrote that there are no solutions to Xn + Yn = Zn for integers X, Y, and Z unless n=2. Less well known is Leonard Euler's generalization of Fermat's theorem. Euler conjectured that an Nth power (N > 2) was never the sum of less than N smaller Nth powers.

In 1966, a computer search found a counterexample to Euler's conjecture. It is that 1445 = 1335 + 1105 + 845 + 275. Since then, no others have been found. The sad fact is: getting computer time at large installations is not easy.

If the search for numerical examples were programmed for the Commodore machines, it is easy to see that running the problem on many machines would give the equivalent of days of time on large IBM, Amdahl, Cray, etc. machines. Suppose 100 Commodore computers devoted 100 hours in 6 months (less than four hours a weekend) to the problem. That is 10,000 hours or, with an improved speed factor of 1,000 for mainframes, 10 hours of equivalent mainframe time for that six month period.

By increasing the number of Commodore 64 and 128 computers working on the project or by increasing the number of hours worked per machine, any speed factor can be dwarfed and many days of equivalent mainframe time can be obtained. I would not be surprised to learn of Commodore computers that can be made available for 100 hours a week. With 100 such machines, we could have ten hours of equivalent mainframe time per week.

The specific task proposed can be separated into smaller tasks. Using 6th powers as an illustration, one computer can look at summing from 1 to 100 as a sixth power, a second at 101 to 120, a third at 121 to 140, and so on for suitable divisions which will lead to approximately equal time to complete. In addition, a search for additional counterexamples to Euler's conjecture must be made for seventh powers, eighth powers, etc. It will not be difficult to set up search lists for 100 computers.

Other tasks scan be tackled in a similar manner. I have been in touch with two eminent mathematicians, Drs. Daniel Shanks and John W. Wrench Jr., who are among those who can suggest other reasonable projects. If Project #1 can be started, I am certain that suggestions for other work will be forthcoming.

To get such a co-operative effort off the ground, several steps are needed. One is to find 6502 machine language programmers who will write efficient code to tackle the problems. A second step is to find an overall project manager, and probably a series of specific project managers. The managers would have the task of seeing to it that the code was written, disks with the projects were prepared and mailed to solvers, or put on bulletin boards. A third step is collecting results, which in most cases will be "no solution found". When these are put together, no one will have to research the same range for the project.

I would appreiate the comments of your readers and of the magazine staff. I have no doubt that the idea is a good one and that it can be improved. Are there people willing to help? Are there computer clubs willing to help? The club could be a specific project's manager as well as a group of solvers. Let them write me or the magazine.

Vincent J. Mooney Jr.
607 Wyngate Drive
Frederick, MD 21701

Very interesting idea, Vincent, though we shudder a bit at the amount of organization it would require. The primary difficulty, once the code was written - which would not too be difficult in the case of the Euler project - would be the assignment of ranges to individuals in such a way as to get exhaustive but not overlapping coverage. The problem is a bureaucratic one and, as with all bureaucratic problems, any solution is going to be time-consuming. Perhaps an on-line service with a lot of subscribers would be the best vehicle for organizing the project, as some kind of rapid, centralized communications facility would probably be requisite if the effort were not to collapse under its own weight. By the way, "Fermat's last theorem" was recently proved (with the aid of computers, I gather), ending at last a couple of centuries of head-scratching. It's amazing that both it and the four-colour theorem have been disposed of in the last ten years. Not to mention the save-@ bug. Do keep us informed. We hope you'll receive an enthusiastic response.

LQ & The Bible: In an old *Transactor* (Letters, Volume 7, Issue 1) magazine, there is an item concerning the data entry of the New Testament. Was this project ever completed? Where can I obtain or purchase a version of the Bible for either a C64 or IBM/XT?

I have another question I'd like to ask. How does one go about building an interface for a true RS-232 port to a C64 serial bus printer? Can such a device be purchased? I would like to use my letter-quality Commodore printer on my IBM/XT.

Garth Usick
Regina, Saskatchewan

Well, Garth, if there is such an interface it has escaped our notice. However, there was (is?) an interface for connecting your CBM serial bus printer to the IBM via the parallel printer port. We seem to remember it as an Omnitronix product. A local BBS user here in Toronto purchased the device and expressed his satisfaction with both the device and the service. Perhaps our readers can provide more information. Biblical text is now available on Commodore disks (as evidenced by the recent ads we've seen since receiving your letter) but we really don't know if this is a product of the project described in Volume 7, Issue 1.

Yet another vote for the ML column!: As a Commodore fan, I would like to know if there are any packages developed, or under development, that make use of the 1764 RAM expansion. Special programs such as RAM disk assemblers and compilers would be a great boost to the 64 programming environment! Before I leave the topic, is the emulation software included truly compatible? Can you run a word processor and tell it to use drive 9 (your RAM disk)? I would also like to add my vote for the assembler subroutine column, since I, like many of your readers, am missing vital routines that must have been written! Writing them again is not very productive.

Your magazine comes out on top in the Commodore world when it comes to good solid information. Keep up the good work!

Amir Michail
Willowdale, Ontario

GEOS uses REUs to a limited extent. The anticipated GEOS upgrade will almost certainly use it to greater advantage. Paperclip III uses the REU for spell checking and Big Blue Reader uses it to buffer file transfers. Most software needs to be rewritten to use the REU although some of the more "well-behaved" ones work with RAMDOS. Disparate data storage methods raise the compatibility question. Check out "On the C Side" in this issue for tips on using the REU with Power C.

REUs and Copyrights: According to the introduction to Dale Castello's article on RAM Expansion Cartridges in the Volume 8, Issue 2, the 1700 and 1750 RAMs work with the C64. (There was also an article in *TPUG Magazine*, Issue 22, by Tim Grantham, where he stated that Commodore Canada had assured him that the 1750 would work with the C64.) Is the reverse true, too? I'm soon going to be moving up to the 128 and if my 1764 RAM will work with it, I would like to stay with it for now. Customer Service at West Chester says it won't work, but then they want to sell more 1750s, don't they.

Also, have any of your hardware hackers come up with a battery-backed system for the 17xx RAMs so they will retain their memory when the system has been powered down (or aren't they the right kind of RAM chips to do this with)?

Now for another query. With all the discussion about copyrights in the T and other magazines, I'm curious about an article I saw in a British magazine, *Commodore Computing International*'s January 1988 issue. The article seems to be a word for word duplication, including the comparison tables, of Mike Garamszeghy's review of the 1581 disk drive in the T (Volume 8, Issue 3). The accompanying program and a paragraph describing it were deleted, but the editors seem to have missed an earlier reference to the program in the body of the text. The article is credited to Mr. Garamszeghy, but there's no mention of the T. The magazine also has no copyright notice with their masthead, as you do.

Also, in the past three issues of *CCI*, there has been a series of Mike's articles on the burst mode that are virtual reprints of his series in *TPUG Magazine* some months ago.

I don't mean to stir up any hornets' nests, but I am curious.

James Greek
New York, New York

Our URS (usually reliable source) tells us that the 1764 will *work with the C128. We have heard of a schematic for adding another 256K to a 1764 but we aren't aware of anyone using a battery-backed REU. It sounds like it could be a very popular hack ,though. In the matter of copyrights, Mike informs us that CCI reprinted the articles with his permission. Since he holds the copyrights on those articles, the decision is his and his alone.*

Answers, anyone?: I recently picked up an Atari joystick for $10 in a closeout. It works fine as a joystick, with the T-J switch flipped to J, but I can't figure out what it does as a track ball, other than mess up the keyboard, making it necessary to read it with interrupts disabled. I wonder if it could be rewired as a 1351 mouse? I can't find the trackball mentioned in any of the Atari literature.

Secondly, does anyone have a working conv52 for the C128 version of C Power? That's the function which converts floats to integers. (I have the Spinnaker "Power C" disk.) The C64 version works fine, but the C128 version gives me random values, based on the fractional part of the number. I wrote to Pro-Line, but the answer I got wasn't very helpful.

Thirdly, is there a version of Buddy (*Transactor*, Volume 8, Issue 4) that does macros, in the sense that I understand macros? The Spinnaker Power Assembler, which is, I gather, the same program, has up to three user-defined pseudo opcodes and three pre-defined multi-instruction pseudo op-codes, but nothing like the usual definition of a macro assembler. To use the user-defined pseudo op-codes, you have to put in extra machine language, with things like "jsr eval", and "jsr put'byte'', or whatever. Usually I think of macros as something like:

```
movd  .mac
lda ?1
sta ?2
lda ?1+1
sta ?2+1
.endmac

movd adr1,adr2
```

where the syntax can vary a bit from assembler to assembler. Maybe I'm splitting hairs, since the Ragsdale assembler which is included with most FORTHs is regarded as a macro assembler.

Finally, I'm mildly surprised that there hasn't been more of a cross-over between 8-bit Commodore types and the Atari ST, on the one hand, and 8-bit Atari types and the Amiga. I should expect to see the people who bought the C64 as a cheap Apple II with improved sound and graphics, much lower price, and, at the time, little available software or support (so you were forced to become a hacker), to go for Tramiel's "Power without the Price". On the other hand, the people who knew the ins and outs of display list interrupt programming on the Atari 800 should now be working with Jay Miner's Amiga chips.

Joel M. Rubin
San Francisco, California

b i t s

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in - if we use it in the bits column, we'll credit you in the column and send you a free one-year's subscription to Transactor

C64 Bits

G-Link on Newer Computers

For all of you G-Link users: our favourite IEEE interface *can* be used with the 64C (even though R44 does not appear on that board). Simply attach the lead on the G-Link to pin 28 of the 6510 in your 64C and you're in business.

Those of you who wish to use the G-Link on a C128 should attach the lead to pin 29 of the 8502. For more info on G-Links, please refer to the Transactor Mail Order section of News BRK.

POKE Poser Figured Out
Randy Thompson, Greensboro, North Carolina

The answer to Vol. 8, Issue 5's "Figure This One Out!" is:

```
1 print "*";: poke 122,0
```

After reading your challenge, the answer was immediately obvious: Simply POKE a zero into the low byte of BASIC's TXTPTR ($7A-$7B) to reset CHRGET. When used in the first line of a BASIC program, this POKE acts as a crude GOTO command.

Congratulations, Randy! You've won the satisfaction of having solved the puzzle. We're still waiting on someone to come up with a second solution. There's a Transactor Bits Book in it for anyone who does.

Self-Save
Ben de Waal, Windhoek, South Africa

After using the Commodore 64 and the 1541, the so-called "save with replace" bug bugged me even in my bed. After reading one of your articles I started to delete my programs before saving them. This was a tedious job because of the length of the delete command. After a few months of doing this, I realized that something had to be done...

SELFSAVE is designed to delete a program before it is saved. By only typing SAVE "filename" the file is first deleted and then SAVEd.

The "SS Creator" program will create a BASIC program called "selfsave" on disk. When RUN, SELFSAVE will transfer 32 bytes of ML to $02A7 and 72 bytes to $A000. The routine needs all of the space at $02A7 (up to $02FF) because the filename is transferred to that location. The code at $A000 is there so that it is not in the way of your other routines. The routine wedges into the SAVE vector and first deletes the file before saving it as normal. This only happens when devices from 8 to 15 are used and wouldn't affect other devices. If the SAVE vector is restored, SYS 679 will direct the vector back again. If you want to disable it, type POKE 818,237: POKE 819,245 and the SAVE vector is back to normal.

```
GK  100 print" ** selfsave - ben de waal  87/12/30 **"
BN  110 n$="selfsave": print "creating the '"n$"' program on disk"
LE  120 nd=200: sa=2049: ch=19472

*** for lines 130-260, see the standard generator program on page 5 ***

AN  1000 data 203,   8,   1,   0, 158,  50,  49,  49,  56,  58, 143,  34
EE  1010 data  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20,  20
KO  1020 data  20,  20,  32,  42,  42,  32,  83,  69,  76,  70,  83,  65
FF  1030 data  86,  69,  32,  45,  32,  66,  69,  78,  32,  68,  69,  32
FF  1040 data  87,  65,  65,  76,  32,  32,  56,  55,  47,  49,  50,  47
EA  1050 data  51,  48,  32,  42,  42,   0,   0,   0,   0, 162,  32, 189
JH  1060 data  95,   8, 157, 167,   2, 202,  16, 247, 162,  73, 189, 127
FF  1070 data   8, 157,   0, 160, 202,  16, 247,  76, 167,   2, 169, 178
DE  1080 data 141,  50,   3, 169,   2, 141,  51,   3,  96, 165,   1,  41
HO  1090 data 254, 133,   1,  32,   0, 160, 165,   1,   9,   1, 133,   1
OG  1100 data  76, 237, 245,  83,  48,  58, 165, 186,  41,   8, 208,   1
PJ  1110 data  96, 162, 183, 181,   0,  72, 232, 224, 189, 208, 248, 165
HF  1120 data 183, 141,  36, 160,  24, 105,   3, 133, 183, 160,   0, 177
EG  1130 data 187, 153, 199,   2, 200, 192,   2, 208, 246, 169, 196, 133
BH  1140 data 187, 169,   2, 133, 188, 169,  15, 168, 166, 186,  32, 186
AP  1150 data 255,  32, 192, 255, 169,  15,  32, 195, 255, 162, 188, 104
EO  1160 data 149,   0, 202, 224, 182, 208, 248,  96
```

RS-232 Bus Shelter

Thomas W. Gurley, Canton, Texas

When the RS-232 channel is either OPENed or CLOSEd, the Kernal ends the routine with CLR. It seems that the programmers at Commodore believed that anyone using the RS-232 bus must be senile and unable to do anything for themselves. That is why the Kernal sets aside the receive and the transmit buffers, does the CLR (because memory was affected) and hopefully selects the correct baud rate for us. The fact that the Kernal sets aside two buffers for use by the RS-232 equipment and clears away variables has presented problems to just about every programmer who has to deal with it. There is an easy fix for both problems.

To help with the problem mentioned by Tony Valeri (Volume 6, Issue 2, p. 48) wherein compiled BASIC programs fail when the RS-232 bus is opened, the programmer can set aside the transmit and the receive buffers just prior to the OPEN statement.

```
100 close 2: rem close always writes a 0 into 248 and 250
110 poke 248,x: poke 250,y
    :rem x=rcv buffer page, y=xmit buffer page
120 open 2,2,2,chr$(a)+chr$(b)+chr$(c)+chr$(d)
    :rem use your usual values here
```

When the Kernal finds a non-zero in 248 and in 250, it skips over the part which sets aside the top of BASIC memory for the buffers. Because the CLOSE routine writes '0' to the buffer pointers, the programmer has to assign the buffers every time after the CLOSE and prior to the OPEN. Even so, when the program is compiled, there is no longer a conflict for the top of memory. You should use a safe memory area above 49152 for your buffers.

One would think that if the Kernal knows when the programmer has taken control and assigned the buffers himself, it would realize that memory was not changed and therefore skip the CLR. But such is not the case. For that, you will have to change the Kernal. It's very easy to do, but the solution cannot be used with existing terminal software.

Those who want to burn their own Kernal into EPROM and who intend to write their own terminal software can take advantage of the simple change, as can those who write the Kernal to the RAM underneath.

Change address 65289 from SEC (56) to CLC (24). This area of the Kernal is common to both OPEN and CLOSE. If the carry is set, a CLR is performed at 57796.

The reason you cannot use this procedure with most existing software is that the buffers *must* be assigned by the program as noted above before OPEN. If this is not done first, the Kernal will assign the buffers to the top of memory. If this is allowed to happen, as it most certainly will with most current software, then the CLR is necessary.

On the other hand, if you have the BASIC version of a terminal program, the change is easy and will allow you to OPEN and CLOSE the RS-232 channel anytime you want without losing variables and without clashing with the compiled program.

Blow Your Stack?

Tony Sultana, Farmers Branch, Texas

Error Check adds a Stack Overflow Error to the list of possible CBM errors. A stack overflow error can occur when too many FOR-NEXT loops or GOSUB routines are nested, or if the stack is too full during an Evaluate Expression (IEVAL) operation (vectored at 778-779).

BASIC stores FOR-NEXT loops and GOSUB routine information on the stack - and IEVAL data temporarily. If less than 62 bytes of storage remains after determining stack space, the BASIC operating system displays an '?out of memory error'. However, such an error can also occur if the BASIC programming space is used up. This short BASIC aid can distinguish between a stack overflow and a real Out of Memory error.

Here's the BASIC loader:

```
MK 10 for x=679 to 747: read a: poke x,a: d=d+a: next
HF 20 if d<>6577 then print"data error": end
PD 30 print"error check activated": sys 679
GC 50 data 173,   0,   3, 141, 200,   2, 173,   1
CE 60 data   3, 141, 201,   2, 169, 195, 141,   0
AF 70 data   3, 169,   2, 141,   1,   3, 169,  96
KG 80 data 141, 167,   2,  96, 224,  16, 240,   3
LN 90 data  76,   0,   0, 138, 186, 228,  34, 144
JE 100 data   3, 170, 176, 244, 169, 221, 160,   2
DF 110 data 32,  30, 171,  76, 101, 164,  83,  84
BB 120 data 65,  67,  75,  32,  79,  86,  69,  82
NK 130 data 70,  76,  79,  87,   0,   0,   0,   0
```

The source code for the stack checking program:

```
        .opt list, gen, noerr
        *=      $02a7
old     = $0000
strout  = $ab1e
error   = $a465
        lda     $0300
        sta     $02c8
        lda     $0301
        sta     $02c9   ;save old vector
        lda     #$c3
        sta     $0300
        lda     #$02
        sta     $0301   ;store new vector
        lda     #$60
        sta     $02a7   ;protect vector
        rts
start   cpx     #$10    ;chk for out of mem
        beq     outmem
```

```
jump    jmp   old
outmem  txa
        tsx
        cpx   $22
        bcc   stackv   ;goto stack overflow
        tax
        bcs   jump
stackv  lda   #$dd
        ldy   #$02
        jsr   strout
        jmp   error
.byte  $53, $54, $41, $43, $4b, $20, $4f, $56
.byte $45, $52, $46, $4c, $4f, $57, $00, $00
```

Data Checker 64
Pontus Lindberg, Veberod, Sweden

This is a useful routine for checking data statements from a long list and also checks for abnormal values (i.e. non-integers and values outside the 0-255 range).

To use it, LOAD it and LIST it. Then LOAD the program to be checked. Now cursor up and hit RETURN on each line of DATA CHECKER. Now RUN it.

Hold down the space bar to scroll. Any abnormal value will be indicated by "error!". Note that if you have typed a lower-case L for a 1 or an upper-case O as a 0, the program will end with a syntax error which will show the line number of the offending DATA element.

```
AO  1 read b: bc=bc+1: rc=rc+1: a=peek(64)*256+peek(63)
MH  2 if b=-1 then print "end of data": goto 9
JM  3 if ch<>a then rc=1: ch=a
BJ  4 if b<>int(b) or b<0 or b>255 then c$="error!"
EA  5 print"line:"a"data"b,c$:c$=""
MM 6 get q$: if q$="" then 6
HG  7 if q$="r"then print bc;rc
NK  8 goto 1
JA  9 end
```

Find Joy
Steven E. Clark, Phoenix, Arizona

Plug joystick into Port One. Be sure joystick is in Port Two. Port One... Port Two...

Are you as tired of the dichotomy as I am? Try the little routine listed below. SYS 828 waits for one of the fire buttons to be pressed, then returns the value of the joystick you used: one or two. You can break out of the wait with a RETURN. When you get back from the routine, you'll find your value stored at 928. If you pressed RETURN, the value will be zero. If you don't want it in the cassette buffer at 828, any location will do. Don't forget to move JOYNUM (the returned value) to a favourite safe location.

One other use of JOYID, with minor changes, might be to start off a program. Instead of 'Press any key to begin', how about 'Press a key or fire button to begin'?

```
NK  10 rem  loader for "joyid1"
PK  20 for i=828 to 916: read x: ch=ch+x: poke i,x: next
AE  30 if ch<>9536 then print"data error": stop
KN  40 print "sys828:peek928 to read joystick number"
HB  100 data 169,   0, 141, 160,   3, 133, 198, 169,   1, 141, 147,   3
JD  110 data 141, 148,   3, 169,  17, 141,  13, 220, 169, 255, 141,   0
JP  120 data 220, 173,   1, 220, 141, 147,   3, 173,   0, 220, 141, 148
LB  130 data   3, 169, 129, 141,  13, 220, 173, 119,   2, 201,  13, 208
NE  140 data   7, 169,   0, 141, 160,   3, 240,  26, 173, 147,   3,  41
MJ  150 data  16, 208,   7, 169,   1, 141, 160,   3, 208,  12, 173, 148
DF  160 data   3,  41,  16, 208, 179, 169,   2, 141, 160,   3, 169,   0
KP  170 data 133, 198,  96,   1,   1
```

Late Night TV
Jason Farah, Davison, Michigan

This is a "dazzler"-type program that simulates a static pattern on a TV set. Make sure the audio is on.

```
NP  10 for n=49152 to 49173
GM 20 read a: poke n,a: next
OJ  30 poke 54273,100: poke 54277,0: poke 54278,255
      : poke 54296,15: poke 54276,129
FB  40 sys 49152
FA  50 data 169, 11, 141, 17, 208, 169, 0, 141, 32, 208, 105
LF  60 data 1, 201, 16, 240, 245, 141, 32, 208, 76, 7, 192
```

Hook, Line and Singer
Chuck Lam, San Francisco, California

Here is an interesting trick for use with the 1660 modem (and maybe other modems with a built-in speaker).

First unplug the telephone cord from the modem and type:

```
poke 56579,peek(56579) or 32:
poke 56577,peek(56577) and 223
```

Now play a music program or any program that uses sound; you should be able to hear the sound from your modem's speaker. Although the sound quality is not really good, it is almost noise free. And at least you know another interesting thing about modems.

After you finish playing with this trick, type:

```
poke 56577,peek(56577) or 32
```

and plug the telephone cord back into the modem.

Note: The above pokes take your modem off-hook, so be *sure* you unplug the telephone cord from the modem.

C-128 Bits

Re-Booting GEOS 128
Richard D. Young, Orleans, Ontario

GEOS 128 functions effortlessly with the 1750 RAM Expansion Unit (REU). Among other things, the REU offers quick and easy re-booting from BASIC, but not without some adjustments. Fast re-boot is one option using the 128 *configure* program in GEOS; if this option has been selected, the 128 will return to GEOS when it is reset. The fastest reset back to GEOS will occur if a copy of the 128 deskTop has been placed in the REU RAM "1571 drive". The GEOS environment will remain reasonably intact, particularly if a copy of *preferences* is also in the RAM drive.

The GEOS manual mentions some conditions that are required before GEOS 128 can be successfully re-booted from BASIC. The most critical of these conditions is that memory in RAM Bank 1 from $C000 to $C07F must remain unmodified. This area of 128 memory is, of course, used by BASIC variables and will be quickly overwritten by strings if a BASIC program is run.

Recognizing this fact, a program called *128 rboot* has been provided with GEOS 128. It provides a clean recovery from RAM Bank 1 changes, when it performs properly. This rboot routine restores Bank 1 at $C000 by FETCHing the required data from the REU. To do this, it must be relocated to an area of common RAM because it must switch to RAM configuration 1 prior to restoring the data. I relocated my version to $0C00 by changing the load address on disk with a disk editor, and changing one absolute address high byte from $1C to $0C. To be safe, I always reset back to GEOS through *128 rboot*.

The easiest way to return to GEOS after running a BASIC program is to include the *128 rboot* routine as DATA statements in the BASIC program, READ and POKE this machine language into memory, and SYS to the re-boot program. The necessary DATA statements can be included as a subroutine; a SYS 3072 will execute the re-boot to GEOS.

I also generally include one more function in any BASIC program I wish to run from the GEOS deskTop. The 1571 disk drive is left in 1541 mode after exiting from GEOS, so I reset it to 1571 mode. One caution: a disk should be inserted in all drives before leaving GEOS.

```
AO 30000 rem reset geos 128 -- ml data for 128 rboot
EM 30010 for i=3072 to 3126: read d: poke i,d: next
FD 30020 return:rem sys 3072 to re-boot geos
EB 30100 data 120, 173,   6, 213,  41,  48,   9,  71
BK 30110 data 141,   6, 213, 169, 126, 141,   0, 255
PE 30120 data 173,  48, 208,  41, 254, 141,  48, 208
HM 30130 data 160,   8, 185,  45,  12, 153,   1, 223
BO 30140 data 136,  16, 247, 173,   0, 233,  41,  64
KM 30150 data 240, 249,  76,   0, 192, 145,   0, 192
IO 30160 data  64, 188,   0, 128,   0,   0,   0
```

Easy 128 Key Fix
Rick Crone, Jackson, Tennessee

My 128 developed a problem with the 'K' key; it would often take two or three strokes to get it to work. Well, soon the aggravation reached critical mass and a solution had to be found. I remembered an article from the T about keyboard repair and searched my back issues.

I found it in Volume 5, Issue 5. So I opened up the 128 and started to follow the instructions. But the 128 had three switches that would require unsoldering (instead of one as in the 64 and PET). Even worse, there were wires running through the back cover of the keyboard, and I couldn't see any obvious way of disconnecting them. I checked the keyboard from the top side and still couldn't see any safe way to get inside.

I pulled the key cap off of the 'K' key and found that with the cap off there is a hole that goes right into the contact area. I used a squirt of cleaner/degreaser (Radio Shack #64-2322 $1.99), put the 128 back together and now the key works great!

I thought this might save some other folks some trouble if they have the same problem with a key. You wouldn't even have to open the case for this repair. I know I sure wouldn't have put up with the aggravation as long as I did if I had known about this quick fix.

Never-never land 128D
John Menke, Mt. Vernon, Illinois

The C128D has a metal chassis. The Cardco?+G printer interface has a power connection that plugs into the cassette port. The connection doesn't fit very well and there's a tendency to fiddle with it despite the exposed template on the top of the connection.

Wrap it with insulating tape or you'll crash the 1571 drive in the C128D. I assure you that sparks do indeed fly when the connector contacts the C128D'S chassis, and the 1571 goes completely off-line (never stops spinning, won't accept commands, 'device not present').

This Bud's for you
Marc Begleiter, Forest Hills, New York

I was having trouble with Buddy-128 when trying to assemble a program with an indirect jump statement. Well I found out what the trouble was! Never include comments on the same line. What appears to be happening is the parser ignores the semicolon and reads the comment as part of the label for the indirect jump. Gee, that was easy. At least it wasn't my fault. Doesn't change my opinion on the assembler though. Love that Bud!

ScrollDir

A scrolling disk directory program for the C-128

by M. Garamszeghy
© 1987 by M. Garamszeghy

The C-128's DIRECTORY or CATALOG command is a vast improvement over the C-64's LOAD "$",8 type of directory. However, it still has some very serious limitations. These include: the inability to obtain a hard copy of the directory without resorting to the LOAD "$",8 method; the inability to scroll the list; and the cumbersome techniques required to LOAD a program or SCRATCH a file directly from the displayed list. If you would like to be able to do these things and more, then this little utility is for you.

SDIR is a memory resident extended directory utility for the C-128 (in 40 or 80 column, FAST or SLOW mode) with a 1541, 1571 or 1581 disk drive. It provides full forward and reverse scrolling capabilities for a directory listing as well as the ability to: provide a hard copy of the directory via a printer; scratch files; load a PRG file; display or merge a SEQ file; change 1581 directory partitions; and validate a disk, all directly from the displayed list.

Creating SDIR

SDIR is written in assembly language using the Buddy-128 system. The source code is some 1000 lines long, and is not included in this article. For those who are interested, it is included on the *Transactor* disk for this issue. Listing 1 is the BASIC loader for the machine language program. Type this in and SAVE it under a name other than "SDIR". Before RUNning the program, you can make changes to the system defaults in lines 1100 to 1170 to reflect your personal set-up. The default values correspond to a disk drive on device 8, an Epson compatible printer as device 4 with a CARDCO interface in transparent mode, and a printed directory listing three entries wide.

The control character values for compressed print on/off and expanded print on/off can be changed to suit your printer. Consult your printer and/or interface manual for details if you are not sure of the appropriate codes. If your printer does not support one of these modes, use a value of 13 (carriage return) or some other harmless value for the applicable parameters. The printer width should be specified in multiples of 32. This parameter divided by 32 will give the number of entries to be printed on a single line. Any value over 64 requires either a wide carriage printer or support for compressed print.

RUN the program once to create the SDIR machine language program. After the program has created the file in memory, you will be prompted to insert a disk into the device 8 disk drive. When the file has been successfully written, you will be asked if you want to start SDIR now. Type in "y <return>" if this is what you wish, or any other response to quit. Once you have created the SDIR file, you no longer need the program in Listing 1 (keep it anyway in case you ever wish to change the default configuration). You can start SDIR on subsequent occasions by the method outlined below.

SDIR Memory Management

The machine language portion of SDIR occupies normally unused BANK 0 RAM between $1300 (decimal 4864) and $1BE0 (decimal 7136). BANK 0 RAM from $D000 upwards is used as the directory buffer. $0B00 to $0DFF (cassette and RS-232 buffers) and $FA to $FF (unused zero page space) are also used as temporary buffers and pointers for various items. These areas are erased and set up each time SDIR is activated.

To prevent BASIC text code from over-writing the machine language portion, the start up routine resets the top of BASIC text limit pointer to $CFFF. This gives over 40K bytes of memory available to BASIC for storing programs and is more than adequate for even the longest of programs. (Remember that on the C-128 variables are stored in BANK 1, and do not take up room in the BASIC work space).

Using SDIR

To start SDIR from disk, the following command is used:

```
BOOT "SDIR"
```

assuming that the machine language program is saved under the name of "sdir". Alternatively with the older C-128 ROM set, SDIR can be activated from the 1541 with:

```
BLOAD "SDIR": SYS 4864
```

Once in memory, the machine language portion of SDIR will remain active until a hard reset is performed on the computer. If it becomes deactivated at any time because the function keys get redefined or the BASIC tokenizer vector at $0304 gets re-set by another utility, SDIR can be restarted by the command:

SYS 4864

The start up routine for SDIR does two main things: it patches itself into the BASIC tokenizing vector and re-defines the F3 key to point to itself rather than the normal BASIC DIRECTORY command. With this patch installed, SDIR becomes a resident command which can be accessed in direct mode only. Of course, DIRECTORY still can be accessed by typing in the command word from BASIC.

The full syntax of the command is:

SD [pattern] [,U<device#>] [,P<printer#>] [,W<printer width>]

All of the parameters are optional and can be specified in any order. The F3 key is redefined as "SD <return>" which works with all defaults. The SD portion of the command line must begin in the first column of a screen line. The optional parameters can be separated by spaces for legibility if desired, although punctuation, etc. is not required.

The pattern can be any legal DOS pattern for directories, including the extended set for the 1581 (only if you're using a 1581 of course). <device#> should be in the range of 8 to 13. An error message will be generated if you try to access a non-existent drive. <printer#> should be 4 or 5. <printer width> is given in number of entires to be printed on a line. It is normally in the range of 2 to 5.

For example, just entering the command "SD" or hitting the F3 key will list all files on the default disk drive (normally device 8) using the default printer and printer width for output.

SD "K*=S",U9,W4,P5

will find all of the SEQ files on device 9 that begin with the letter K. If a printout is selected later, it will be given on the device 5 printer at 4 entries per line.

The simplest way to use SDIR is to just put a disk into your drive and press the F3 key - the F3 key was chosen for this task because its default definition in BASIC 7.0 is DIRECTORY. Alternatively, you can enter the SD command along with its optional parameters described above.

After a few seconds, the disk directory will be printed on the screen. If you are using an 80 column screen, a command summary will be printed on the right hand side of the screen. No command summary is provided on the 40 column screen due to space considerations. A quick summary is given in Table 1.

The directory listing takes the following format:

filename type size

The type will be one of PRG, USR, REL, SEQ or CBM (1581 only). Locked "<" and splat "*" status are also indicated. The file size is given in blocks. The disk name, number of blocks free and number of files listed is also displayed. Up to 20 files can be displayed on the screen at one time. The following command options are possible:

• Use the <cursor up> and <cursor down> keys to scroll through the displayed list. The currently selected file will be highlighted in reverse video.

• The <home> key will take you back to the top of the list.

• The <esc> key will clear the screen and go back to BASIC.

• The logo-p key combination (i.e. hold down the Commodore logo key at the lower left corner of the keyboard and the"p" key simultaneously) will give a hard copy of the entire directory on a printer and return to the SDIR display. If supported by your printer, the disk name and ID code will be printed in double width, while the entries will be in compressed print. The number of files found and blocks free will be printed in normal size.

• The <return> key has three functions, depending on the file type. For PRG files, it acts like a BLOAD command and will automatically load the highlighted file. Be careful with BASIC programs: make sure that the graphics screen allocation state is the same as when the program was saved. (If you BLOAD a BASIC program that was saved when the graphics screen was allocated, it comes from a start of BASIC address of $4000, rather than the normal start of BASIC address of $1C00.)

For a 1581 CBM directory partition file, <return> will switch the current partition to the selected file.

For other file types, <return> will display the contents of the file on the screen then return to the SDIR menu. This will not affect any BASIC program that may be in memory. Press the <run/stop> key to abort a file display if you decide that you do not want to view the entire file. The <no scroll> key will pause the display momentarily until another key is pressed.

• The key combination logo-m will cause a SEQ program file listing to be MERGEd with any BASIC program currently in memory. A listing can be created with the simple command sequence:

OPEN 8,8,8,"PROGRAM.LIST,S,W": CMD 8: LIST
PRINT#8: CLOSE8

A SEQ program listing is also sometimes used for downloading files from bulletin board systems. The logo-m command will automatically re-crunch the file into PRG format. After

the MERGE has been completed (usually by the printing of an '?out of data' error on the screen), you must type in CLOSE#1 to close the disk file. (The '?out of data' error is caused by the "READY." message which is included at the end of every Commodore BASIC listing. The computer interprets this as READ Y. Since no DATA statements are included, you get the '?out of data' message). Logo-m can also be used to execute a sequential disk command file as outlined in *Transactor* Volume 8, Issue 2 ("SYS 65478 revisited" on page 33).

• The key combination logo-r will return a 1581 to its root directory partition and initialize the drive. For 1541 or 1571 drives, it just initializes the drive ("I0"). For all drives, it will also select the full directory if a pattern was originally specified.

• The logo-s key combination will scratch the selected file. Be careful when you use this, because *you are not prompted to confirm your request to delete the file!* Once the file is gone, it is *gone* (unless you fix the disk with a sector editor). After deletion, SDIR will re-read the directory using the original pattern.

• The logo-v key will perform a disk validation, then re-read the directory.

Final Observations

Unlike most programs that deal with disk files, SDIR credits the user with a degree of intelligence. Although it has extensive error detection routines, you will not be prompted or cajoled "are you sure?" each time you press a key. Because of this, a certain amount of caution may be required, especially when scratching files. Otherwise, SDIR is much faster for people who are relatively careful.

Table 1: SDIR Quick Command Reference

Command	Action
<cursor up>	Scroll up list
<cursor dn>	Scroll down list
<home>	Go to top of list
<esc>	Exit to Basic
<return>	BLOAD PRG file
	Set 1581 directory
	Display SEQ file
C= m	Merge SEQ file
C= p	Print directory list
C= r	Set 1581 root dir
	Reset dir pattern to all
C= s	Scratch file
C= v	Validate disk

Listing 1: BASIC program to create the "SDIR" machine language program on disk.

```
OC  1000 rem*****************************
CI  1010 rem*       sdir 4.0            *
PH  1020 rem*   by m. garamszeghy       *
PE  1030 rem*       87-09-01            *
GF  1040 rem*****************************
OI  1050 :
BI  1060 cs=0: bank 0: print "working ..."
CA  1070 for i=4864 to 7126: read x: cs=cs+x: poke i,x: next
EF  1080 if cs<>222651 then print "error in data statements": end
GL  1090 :
FM  1100 poke 4867,8  : rem default disk drive device#
OG  1110 poke 4868,4  : rem default printer device#
JM  1120 poke 4869,4  : rem default printer sec address
FO  1130 poke 4870,96 : rem default # printer columns per page
JJ  1140 poke 4871,15 : rem printer code to set compressed print
NH  1150 poke 4872,18 : rem printer code to cancel compressed print
JG  1160 poke 4873,14 : rem printer code for expanded print
DE  1170 poke 4874,20 : rem printer code to cancel expanded print
AB  1180 :
HB  1200 print"insert disk then press a key to continue..."
PA  1210 getkey a$
DA  1220 bsave"sdir",b0,p4864 to p7136: if ds then print ds$: end
HG  1230 print "--> sdir4 file created <--": bank 15
LH  1240 input"start sdir <y/n>";ss$
JO  1250 if ss$="y" then sys 4864
MO  1260 end
OJ  2000 data  76,  11,  19,   8,   4,   4,  96,  15
LP  2010 data  18,  14,  20, 162,   0, 134, 252,  32
GE  2020 data 221,  26,  32, 101,  19, 169, 207, 141
NC  2030 data  19,  18, 169, 255, 141,  18,  18, 169
LK  2040 data 154, 141,   4,   3, 169,  19, 141,   5
IH  2050 data   3,  32, 125, 255,  13,  13,  83,  89
IJ  2060 data  78,  84,  65,  88,  58,  32,  32,  83
AJ  2070 data  68,  32,  34,  80,  65,  84,  84,  69
CJ  2080 data  82,  78,  34,  44,  85,  60,  68,  69
KE  2090 data  86,  73,  67,  69,  35,  62,  13,   0
LK  2100 data 169,  97, 133, 250, 169,  19, 133, 251
OD  2110 data 169, 250, 160,   4, 162,   3,  76, 101
OF  2120 data 255,  13,  83,  68,  13,  32, 125, 255
AL  2130 data  83,  68,  73,  82,  32,  52,  46,  48
GK  2140 data  32,  32,  60,  67,  62,  49,  57,  56
CL  2150 data  55,  32,  77,  46,  32,  71,  65,  82
EA  2160 data  65,  77,  83,  90,  69,  71,  72,  89
AI  2170 data   0,  96, 169,   0, 141,   0, 255,  32
PG  2180 data 231, 255, 162,   0,  76, 221,  26,  36
DB  2190 data  48,  58, 173,   0,   2, 201,  83, 208
FF  2200 data   7, 173,   1,   2, 201,  68, 240,   3
JD  2210 data  76,  13,  67, 169,   0, 141,   0, 255
BK  2220 data 168, 153,   0,  13, 200, 208, 250,  32
MO  2230 data 204, 255,  32, 231, 255, 162,   0,  32
MP  2240 data 221,  26, 162,   3,  32, 221,  26,  32
HN  2250 data 101,  19, 162,   1,  32, 221,  26, 173
AM  2260 data   3,  19, 141,   5,  13, 173,   6,  19
DO  2270 data 141,   4,  13, 173,   4,  19, 141,   2
FG  2280 data  13, 173,   5,  19, 141,   3,  13, 160
FK  2290 data   0, 185, 151,  19, 153,  32,  13, 200
DP  2300 data 192,   3, 208, 245, 136, 140,  13,  13
```

```
PO  2310 data 160,   0, 185,   0,   2, 208,   3,  76
LA  2320 data 131,  20, 201,  34, 240,  86, 201,  85
GB  2330 data 240,  13, 201,  80, 240,  22, 201,  87
BH  2340 data 240,  31, 200, 208, 229, 240, 108,  32
KK  2350 data  72,  20, 208,   3, 173,   3,  19, 141
AF  2360 data   5,  13, 208, 238,  32,  72,  20, 208
LF  2370 data   3, 173,   4,  19, 141,   2,  13, 208
NG  2380 data 225,  32,  72,  20,  41,   7, 170, 189
OG  2390 data  64,  20, 141,   4,  13,  76,  18,  20
BK  2400 data  64,  64,  64,  96, 128, 160, 192,  64
LF  2410 data 200, 185,   0,   2,  41,  15, 201,   1
BD  2420 data 208,   9, 200, 185,   0,   2,  41,   3
PF  2430 data  24, 105,  10,  96, 162,   0, 200, 185
KL  2440 data   0,   2, 141,  20,  13, 240,  10, 201
NJ  2450 data  34, 240,   6, 157,  35,  13, 232, 208
OH  2460 data 237, 224,   0, 240,  14, 232, 138,  24
MF  2470 data 109,  13,  13, 141,  13,  13, 173,  20
NF  2480 data  13, 208, 143, 169,   0, 141,   0,   2
NA  2490 data 169,  14, 174,   5,  13, 160,  15,  32
EP  2500 data 186, 255, 169,   0,  32, 189, 255,  32
BA  2510 data 192, 255, 144,   3,  76,  14,  27, 169
PI  2520 data  73, 141,   0,  12, 169,  48, 141,   1
EA  2530 data  12, 169,   2, 141,  10,  13,  32, 117
JC  2540 data  27, 173,  12,  13, 240,   3,  76, 138
JH  2550 data  19,  32,  87,  27, 173,   0,  12, 201
GL  2560 data  48, 240,   6,  32,  14,  27,  76, 138
HB  2570 data  19,  32, 194,  26, 162,   1,  32, 221
PG  2580 data  26,  32, 125, 255,  87,  79,  82,  75
II  2590 data  73,  78,  71,  46,  46,  46,   0, 169
HE  2600 data   1, 174,   5,  13, 160,   0,  32, 186
CD  2610 data 255, 173,  13,  13, 162,  32, 160,  13
AK  2620 data  32, 189, 255, 169,   0, 170,  32, 104
DK  2630 data 255,  32, 192, 255,  32,  87,  27, 173
PF  2640 data   0,  12, 201,  48, 208, 189, 162,   1
AK  2650 data  32, 198, 255, 160,   0, 140,   0,  13
KP  2660 data 140,   1,  13,  32, 148,  27, 144,   3
GJ  2670 data  76, 167,  21, 201,  34, 208, 244,  32
NP  2680 data 148,  27, 201,  34, 240, 249, 153,  96
EP  2690 data  13, 200, 192,  22, 208, 241, 169,  32
GG  2700 data 160,   0,  32, 159,  27, 200, 192,  32
FL  2710 data 208, 248,  32, 148,  27, 144,   3,  76
JO  2720 data 167,  21, 208, 246,  32, 148,  27,  32
LA  2730 data 148,  27, 144,   3,  76, 167,  21,  32
JL  2740 data 148,  27, 141,  22,  13,  32, 148,  27
AK  2750 data 174,  22,  13,  32, 207,  26, 160,  22
LA  2760 data 185, 234,   0, 240,   6,  32, 159,  27
IM  2770 data 200, 208, 245, 160,   0,  32, 148,  27
GM  2780 data 144,   3,  76, 167,  21, 201,  34, 208
HC  2790 data 244,  32, 148,  27, 144,   3,  76, 167
LN  2800 data  21, 201,  34, 240, 244,  32, 159,  27
GK  2810 data 200, 192,  22, 208, 236,  32, 185,  27
BC  2820 data 238,   0,  13, 208,   3, 238,   1,  13
AK  2830 data  76,  46,  21,  66,  76,  79,  67,  75
BI  2840 data  83,  32,  70,  82,  69,  69,  32,  32
LI  2850 data 204, 255, 169,   1,  32, 195, 255, 160
LA  2860 data  22,  32, 172,  27, 153, 162,  13, 200
LO  2870 data 192,  32, 208, 245, 160,   0, 185, 155
GM  2880 data  21, 153, 189,  13, 200, 192,  12, 208
CH  2890 data 245, 169, 255, 160,   0,  32, 159,  27
FG  2900 data 162,   3,  32, 221,  26,  32, 101,  19
JG  2910 data 162,   4,  32, 221,  26, 160,   0, 185
IG  2920 data  96,  13,  32, 210, 255, 200, 192,  24
HH  2930 data 208, 245, 173,   1,  13, 174,   0,  13
BN  2940 data  32, 207,  26, 160,   0,  32, 197,  27
GP  2950 data 185,   0,   1, 240,   9, 153, 160,  13
NC  2960 data  32, 210, 255, 200, 208, 242, 160,   0
NJ  2970 data 185,  22,  22, 240,  21,  32, 210, 255
GD  2980 data 153, 165,  13, 200, 208, 242,  32,  32
BP  2990 data  70,  73,  76,  69,  83,  32,  32,  32
CI  3000 data  32,   0, 168, 185, 184,  13,  32, 210
HJ  3010 data 255, 200, 192,  18, 208, 245, 162,   1
OD  3020 data 173,   0,  13, 208,  33, 173,   1,  13
IO  3030 data 208,  28,  32, 221,  26,  32, 125, 255
LH  3040 data  78,  79,  32,  70,  73,  76,  69,  83
FB  3050 data  32,  70,  79,  85,  78,  68,  13,   0
FD  3060 data  32,  59,  27,  76, 138,  19, 232, 165
EG  3070 data 215, 208,   3,  76,   4,  24,  32, 221
PB  3080 data  26,  32, 125, 255, 176, 192, 192, 192
IM  3090 data 192, 192, 192, 192, 192, 192, 192, 192
CN  3100 data 192, 192, 192, 192, 192, 192, 192, 192
MN  3110 data 192, 192, 192, 192, 192, 192, 192, 192
DI  3120 data 192, 192, 174,  13, 221,  32,  60,  85
GJ  3130 data  80,  62,  44,  60,  68,  78,  62,  32
JL  3140 data  45,  32,  83,  67,  82,  79,  76,  76
GJ  3150 data  32,  76,  73,  83,  84,  32,  32,  32
KE  3160 data  32,  32, 221,  13, 221,  32,  32,  32
NL  3170 data  32,  60,  72,  79,  77,  69,  62,  32
EM  3180 data  45,  32,  84,  79,  80,  32,  79,  70
OL  3190 data  32,  76,  73,  83,  84,  32,  32,  32
CH  3200 data  32,  32, 221,  13, 221,  32,  32,  32
JM  3210 data  32,  32,  60,  69,  83,  67,  62,  32
GA  3220 data  45,  32,  69,  88,  73,  84,  32,  84
NA  3230 data  79,  32,  66,  65,  83,  73,  67,  32
KJ  3240 data  32,  32, 221,  13, 221,  32,  32,  60
KC  3250 data  82,  69,  84,  85,  82,  78,  62,  32
CD  3260 data  45,  32,  66,  76,  79,  65,  68,  32
LN  3270 data  80,  82,  71,  32,  32,  32,  32,  32
CM  3280 data  32,  32, 221,  13, 221,  32,  32,  32
MN  3290 data  32,  32,  32,  32,  32,  32,  32,  32
PE  3300 data  45,  32,  82,  69,  65,  68,  32,  83
MB  3310 data  69,  81,  32,  32,  32,  32,  32,  32
KO  3320 data  32,  32, 221,  13, 221,  32,  32,  32
EA  3330 data  32,  32,  32,  32,  32,  32,  32,  32
OH  3340 data  45,  32,  67,  72,  65,  78,  71,  69
OH  3350 data  32,  49,  53,  56,  49,  32,  68,  73
BC  3360 data  82,  32, 221,  13, 221,  32,  32,  32
AF  3370 data  32,  32,  32,  67,  61,  32,  77,  32
LI  3380 data  45,  32,  77,  69,  82,  71,  69,  32
BH  3390 data  83,  69,  81,  32,  32,  32,  32,  32
KD  3400 data  32,  32, 221,  13, 221,  32,  32,  32
LG  3410 data  32,  32,  32,  67,  61,  32,  80,  32
FL  3420 data  45,  32,  80,  82,  73,  78,  84,  32
JJ  3430 data  68,  73,  82,  32,  32,  32,  32,  32
CG  3440 data  32,  32, 221,  13, 221,  32,  32,  32
HJ  3450 data  32,  32,  32,  67,  61,  32,  82,  32
OO  3460 data  45,  32,  49,  53,  56,  49,  32,  82
PA  3470 data  79,  79,  84,  32,  68,  73,  82,  32
KI  3480 data  32,  32, 221,  13, 221,  32,  32,  32
BM  3490 data  32,  32,  32,  67,  61,  32,  83,  32
NA  3500 data  45,  32,  83,  67,  82,  65,  84,  67
MA  3510 data  72,  32,  70,  73,  76,  69,  32,  32
CL  3520 data  32,  32, 221,  13, 221,  32,  32,  32
PO  3530 data  32,  32,  32,  67,  61,  32,  86,  32
GE  3540 data  45,  32,  86,  65,  76,  73,  68,  65
LE  3550 data  84,  69,  32,  68,  73,  83,  75,  32
LC  3560 data  32,  32, 221,  13, 173, 192, 192, 192
```

```
IK 3570 data 192, 192, 192, 192, 192, 192, 192, 192
CL 3580 data 192, 192, 192, 192, 192, 192, 192, 192
ML 3590 data 192, 192, 192, 192, 192, 192, 192, 192
DF 3600 data 192, 192, 189,   0,  32, 194,  26, 162
JG 3610 data   0, 142,  18,  13, 142,   6,  13, 142
GI 3620 data   7,  13, 142,   8,  13, 232,  32, 221
KE 3630 data  26, 160,   0,  32, 172,  27, 201, 255
FE 3640 data 208,   3,  76,  64,  24,  32, 210, 255
OA 3650 data 200, 192,  28, 208, 238,  32, 185,  27
PB 3660 data 238,  18,  13, 173,  18,  13, 201,  20
DP 3670 data 240,   6,  32, 197,  27,  76,  25,  24
BM 3680 data  32, 194,  26,  32,  73,  24,  76, 146
BF 3690 data  24, 169,  18,  32, 210, 255, 160,   0
GH 3700 data 174,   6,  13,  24,  32, 240, 255, 160
AE 3710 data   0,  32, 172,  27,  32, 210, 255, 200
CB 3720 data 192,  31, 208, 245, 169, 146,  76, 210
IM 3730 data 255,  32,  78,  24, 174,   7,  13, 232
OK 3740 data 236,   0,  13, 208,   3,  76,  73,  24
EC 3750 data 142,   7,  13, 174,   6,  13, 224,  19
IL 3760 data 240,   7, 232, 142,   6,  13,  76, 140
OB 3770 data  24,  32, 197,  27,  32, 185,  27,  76
IC 3780 data  73,  24,  32, 228, 255, 240, 251, 141
MD 3790 data   9,  13, 201,  27, 208,   3,  76, 138
IC 3800 data  19, 201,  19, 208,   3,  76,   4,  24
DC 3810 data 201,  13, 208,   3,  76,  38,  25, 201
DP 3820 data 167, 208,   3,  76,  38,  25, 201, 175
GO 3830 data 208,   3,  76,  34,  26, 201, 145, 240
JF 3840 data  36, 201,  17, 208,   3,  76, 235,  24
KO 3850 data 201, 174, 208,   3,  76,  38,  25, 201
HD 3860 data 178, 208,   8, 162,   2, 142,  13,  13
MC 3870 data  76, 159,  20, 201, 190, 208,   3,  76
GC 3880 data 143,  25,  76, 146,  24,  32, 241,  24
DG 3890 data  76, 146,  24,  32, 105,  24,  76, 146
GM 3900 data  24,  32,  78,  24, 174,   7,  13, 202
CE 3910 data 224, 255, 208,   3,  76,  73,  24, 142
NE 3920 data   7,  13, 174,   6,  13, 240,   7, 202
BA 3930 data 142,   6,  13,  76,  24,  25, 169,  27
HA 3940 data  32, 210, 255, 169,  73,  32, 210, 255
GL 3950 data  56, 165, 252, 233,  32, 133, 252, 176
IL 3960 data   2, 198, 253,  76,  73,  24, 160,  16
NI 3970 data  32, 172,  27,  41, 127, 201,  32, 208
HF 3980 data   3, 136, 208, 244, 200, 140,  16,  13
KK 3990 data 173,   9,  13, 201, 174, 208,  16, 160
LH 4000 data   0, 185, 137,  25, 153,   0,  12, 200
GM 4010 data 192,   3, 208, 245,  76, 105,  25, 160
GL 4020 data  17,  32, 172,  27, 201,  67, 208,  77
BL 4030 data 160,   0, 185, 134,  25, 153,   0,  12
CC 4040 data 200, 192,   3, 208, 245, 136, 140,  13
DI 4050 data  13, 160,   0,  32, 172,  27, 153,   3
IN 4060 data  12, 200, 204,  16,  13, 208, 244,  24
JP 4070 data 173,  16,  13, 105,   3, 141,  10,  13
CE 4080 data  32, 117,  27,  76, 201,  20,  47,  48
PN 4090 data  58,  83,  48,  58,  86,  48,  58, 160
MM 4100 data   0, 185, 140,  25, 153,   0,  12, 200
PM 4110 data 192,   3, 208, 245, 140,  10,  13,  32
HB 4120 data 117,  27,  76, 201,  20, 162,   0,  32
CG 4130 data 221,  26, 169,   1, 174,   5,  13, 160
LA 4140 data   3,  32, 186, 255, 173,  16,  13, 166
KD 4150 data 252, 164, 253,  32, 189, 255, 169,   0
MN 4160 data 170,  32, 104, 255, 160,  17,  32, 172
IG 4170 data  27, 201,  80, 208,  23,  24, 169,   0
LH 4180 data  32, 213, 255, 142,  16,  18, 140,  17
GE 4190 data  18, 176,   3,  76, 138,  19,  32,  87
NJ 4200 data  27,  76,   4,  24,  32, 192, 255, 162
HH 4210 data   1,  32, 198, 255, 173,   9,  13, 201
NI 4220 data 167, 208,  12, 160,  17,  32, 172,  27
PO 4230 data 201,  83, 208,   3,  76, 146,  19,  32
BL 4240 data 207, 255,  32, 210, 255,  32, 225, 255
NK 4250 data  16,   5,  32, 183, 255, 240, 240,  32
BL 4260 data 204, 255, 169,   1,  32, 195, 255,  32
HF 4270 data  59,  27, 162,   0,  32, 221,  26,  76
JO 4280 data 208,  21, 169,   6, 174,   2,  13, 172
NE 4290 data   3,  13,  32, 186, 255, 169,   0,  32
HC 4300 data 189, 255,  32, 192, 255,  24, 162,   6
KA 4310 data  32, 201, 255, 144,   3,  76, 183,  26
OB 4320 data  32, 194,  26, 173,   9,  19,  32, 210
GM 4330 data 255, 160,   0, 185,  96,  13,  32, 210
BF 4340 data 255, 200, 192,  32, 208, 245,  32, 197
KD 4350 data  27,  32, 197,  27, 173,  10,  19,  32
ED 4360 data 210, 255, 173,   7,  19,  32, 210, 255
AE 4370 data 172,   4,  13,  32, 202,  27,  32, 197
MD 4380 data  27, 160,   0,  32, 172,  27, 201, 255
HA 4390 data 240,  27,  32, 210, 255, 200, 204,   4
JG 4400 data  13, 208, 240,  32, 197,  27,  24, 165
NI 4410 data 252, 109,   4,  13, 133, 252, 144, 225
OH 4420 data 230, 253,  76, 113,  26,  32, 197,  27
MH 4430 data 172,   4,  13,  32, 202,  27,  32, 197
LH 4440 data  27, 173,   8,  19,  32, 210, 255, 160
FE 4450 data   0, 185, 160,  13,  32, 210, 255, 200
BO 4460 data 192,  48, 208, 245,  32, 197,  27,  32
GK 4470 data 204, 255, 169,   6,  32, 195, 255,  76
MH 4480 data   4,  24, 169,   0, 133, 252, 133, 254
DM 4490 data 169, 192, 133, 253, 133, 255,  96, 133
HK 4500 data 100, 134, 101, 162, 144,  56,  32, 117
JL 4510 data 140,  32,  68, 142,  96, 189, 250,  26
AP 4520 data 133, 228, 189, 255,  26, 133, 229, 165
FO 4530 data 215, 240,  10, 189,   4,  27, 133, 230
DP 4540 data 189,   9,  27, 133, 231, 169, 147,  76
JM 4550 data 210, 255,  24,  22,  24,  24,   2,   0
PN 4560 data   3,   5,  24,   0,   0,   0,  45,   0
CB 4570 data   0,  79,  44,  79,  44,  44,  32, 204
DM 4580 data 255, 162,   1,  32, 221,  26,  32, 125
IL 4590 data 255,  13,  13,  69,  82,  82,  79,  82
HH 4600 data  58,  13,   0, 173,  11,  13, 240,  14
FP 4610 data 160,   0, 185,   0,  12,  32, 210, 255
GG 4620 data 200, 204,  11,  13, 208, 244, 169,  64
CN 4630 data 141,  12,  13,  32, 125, 255,  13,  13
FH 4640 data  80,  82,  69,  83,  83,  32,  65,  32
FJ 4650 data  75,  69,  89,  32,  46,  46,  46,  13
JA 4660 data   0,  32, 228, 255, 240, 251,  96,  24
GG 4670 data 162,  14,  32, 198, 255, 176, 175, 160
KB 4680 data   0,  32, 207, 255, 153,   0,  12, 201
ID 4690 data  13, 240,   3, 200, 208, 243, 140,  11
DC 4700 data  13,  32, 204, 255,  96, 162,  14,  24
PK 4710 data  32, 201, 255, 176,  17, 160,   0, 185
BB 4720 data   0,  12,  32, 210, 255, 200, 204,  10
LA 4730 data  13, 208, 244,  76, 204, 255,  32, 204
EM 4740 data 255,  76,  14,  27,  56,  32, 183, 255
BJ 4750 data 208,   4,  24,  32, 207, 255,  96, 162
GG 4760 data  63, 142,   0, 255, 145, 252, 162,   0
LK 4770 data 142,   0, 255,  96, 162,  63, 142,   0
GN 4780 data 255, 177, 252, 162,   0, 142,   0, 255
GC 4790 data  96,  24, 165, 252, 105,  32, 133, 252
NJ 4800 data 144,   2, 230, 253,  96, 169,  13,  76
LN 4810 data 210, 255, 169,  61,  32, 210, 255, 136
IJ 4820 data 208, 248,  96, 255,   0,   0,   0
```

Multitasking on the Commodore 128

Mysterious force or simple programming trick?

by Mike Mohilo

Multitasking is really a mysterious force that only inhabits computers like the Amiga, or is it? Actually it is just a simple programming trick that can even be done on the Commodore 128. This program will allow up to four different programs to run at the same time provided that they don't interfere with each other. A program doesn't have to be an IRQ routine to run in the background. Anything that ends with an RTS or even the monitor can be run in the background. For example, the first demo program (MULTI.B1) will let you have full use of the monitor while BASIC runs a short program. Imagine using the monitor to debug a program while it is running! Have you ever wanted the power to switch from a word processor to a spreadsheet or to BASIC and back again without saving several files and swapping just as many disks? For a demonstration of the idea, run MULTI.B2 and you will be able to switch from the monitor to BASIC even if a BASIC program is running and there is no cursor.

General operation

Getting things started is fairly straightforward. The multitasking program and any other programs are loaded. The initialization routine is called first. Afterwards, background tasks are created with another set of subroutines. This simply tells the multitasking program where it can find your programs. Programs that have been entered into the multitasker can now be told to run or stop with either a subroutine or directly from the keyboard.

Each background task is assigned a number and a key. Task #0 (which is usually BASIC) is switched on or off by pressing [SHIFT][RESTORE]. Task #1 is switched by [C=][RESTORE], #2 is switched by [ALT][RESTORE]. Reading these keys from the NMI routine triggered by the RESTORE key probably won't interfere with your programs. Any combination of the four available tasks can be toggled on or off by hitting the appropriate keys.

In some cases, having more than one task running at a time would be undesirable, so an OTAT (one task at a time) mode is included. For example, when task #2 is turned on, tasks 0, 1, and 3 are turned off and kept out of the way. Another option will display the status of all tasks whenever one is selected with the restore key. The restore key routine is idiot proof and it will prevent everything from being turned off or a non-existent task from being turned on. [RUN/STOP][RESTORE] is not affected by the program.

Initialization

The INIT routine at $1300 sets the IRQ and NMI vectors and starts the multitasker. To display task status when a task is toggled with the restore key, set the accumulator to 1. To allow only one task to run at a time, set the X register to 1. This routine can be run at any time without disturbing background tasks.

Creating a Task

Three routines are used to define a background task. SETREGS at $1303 will set the A, X, Y, and P registers of a new task. SETPROG at $1306 sets the bank and starting address of the program to be run. The bank value is stored in A, low byte of start address in X, and high byte in Y. Note that the bank value is poked directly into the MMU at $FF00. The CREATE routine at $1309 will create the new task by preparing a stack for it and recording it in a task table. The task number is stored in A and task #0 does not need to be created since it exists at the moment you turn on the computer. SETREGS and SETPROG must be used before CREATE and they will not affect a previously created task.

Using the runstop routine

The RUNSTOP routine at $130C can give absolute control over a task regardless of the restrictions on the restore key routine. It can even turn every task off (a bad idea since it crashes the machine). The task number is stored in Y and run/stop is stored in X as a 0 or a 1. A program can get absolute priority and run uninterrupted by the multitasker if this routine is called with the carry bit set. It will disable the multitasker but not the normal IRQ until it is called again with the carry bit

clear. Unimportant background tasks can be slowed down by setting a delay value greater than 0 in the accumulator (see MULTI.B1 for an example).

Kill and load/save

Programs that terminate with an RTS will automatically return to the KILL routine. The return address to KILL was placed on the stack by CREATE. Note that task #0 was not made with CREATE so it will not return to KILL. To prevent a collision between the Kernal load, save, and other I/O, load and save are trapped and run with the priority mode set (see RUNSTOP). This allows them to run without interference.

The IRQ routine

During an IRQ, all of the registers including the bank are stored on the stack. After all of the IRQ work is done, all of the registers are put back and the program that was interrupted runs as if nothing had happened. To perform multitasking, the IRQ sequence runs normally until the end, when registers for a different program are put back. With each IRQ, one program's registers are stored and another's are put back, causing each program to run for a brief moment between IRQs. This happens quickly enough that all programs appear to run at the same time. Since the registers are stored on the stack, several sets of them can be stored simply by switching between several stacks. The MMU chip has an interesting feature that can relocate the stack or zero page to any convenient place. Switching from one program to the next is simply a matter of switching from one stack to another. The newly installed IRQ routine switches stacks and stack pointers according to a list of available background tasks - the very same list made by the CREATE and RUNSTOP routines. The entire process is very fast and the background tasks are completely unaware of what happened.

Here is a more detailed description of what happens during the IRQ. The IRQ signal to the microprocessor from one of the I/O chips starts the process every 1/60th of a second. First, the status register and a return address from the interrupted program are put on the stack. The Kernal IRQ routine is entered. This puts the A, X, Y, and MMU configuration on the stack. The status register is tested to see if a BRK instruction caused the interrupt. At this point the IRQ can be trapped and made to do as I wish. Normal housekeeping is done (scan keyboard, update clock, etc.). Now I find a new task to run and change the stack and stack pointer acordingly. The time delay function to slow a program down works here too. Now that all housekeeping and task swapping is done, it is time to put the bank, A, X, Y, status, and return address back where they belong (the RTI instruction does some of this). Now the interrupted program is back and running.

Unfortunately the Commodore 128 wasn't designed to be a real multitasking machine and without careful planning, use of the Kernal I/O routines by several tasks at a time will cause bad things to happen. Maybe someone can fix this?

Listing 1: multi.B1. This program uses multitasking to allow a BASIC program to run while you use the ML monitor.

```
MB  10 rem ***** multi.b1 *****
ML  20 rem basic on/off   -   shift-restore
BF  30 rem monitor on/off -   logo-restore
IC  40 fast: scnclr: bank15: bload"multi.ml"
OH  50 sys 4864,1,0 :rem init -display tasks -on/off toggle
PI  60 sys 4867,0,0,0,0     :rem  set a,x,y,p
GH  70 sys 4870,0,0,176     :rem  set bank15 and $b000
NN  80 sys 4873,1           :rem  create task#1 (monitor)
KH  90 sys 4876,0,1,1,0     :rem  start task#1
CJ  100 play"cdefgab"
JA  110 sys 4876,100,1,0,0  :rem  set delay=100  task#0
GK  120 play"cdefgab"
KL  130 sys 4876,0,1,0,0    :rem  set no delay (basic)
BC  140 goto 100
```

Listing 2: multi.B2. This example uses the one-task-at-a-time mode to allow switching between BASIC and the monitor.

```
PB  10 rem ***** multi.b2 *****
CI  20 rem switch to basic   -  shift-restore
GP  30 rem          monitor -  logo-restore
IC  40 fast: scnclr: bank15: bload"multi.ml"
EJ  50 sys 4864,0,1 :rem init -no display -one task at time
PI  60 sys 4867,0,0,0,0  :rem  set a,x,y,p
GH  70 sys 4870,0,0,176  :rem  set bank15 and $b000
NN  80 sys 4873,1        :rem  create task#1 (monitor)
```

Listing 3: BASIC generator program for the multitasking system. This will create the file "multi.ml" on disk.

```
PI  1000 rem  generator for "multi.ml"
EN  1010 nd$="multi.ml": rem  name of program
IA  1020 nd=529: sa=4864: ch=57790
OG  1030 for i=1 to nd: read x
IK  1040 ch=ch-x: next
ML  1050 if ch<>0 then print"data error": stop
JN  1060 print"data ok, now creating file": print
GE  1070 restore
LP  1080 open 8,8,1,"0:"+f$
GN  1090 print#8,chr$(sa/256)chr$(sa-int(sa/256));
EL  1100 for i=1 to nd: read x
GN  1110 print#8,chr$(x);: next
IE  1120 close 8
GG  1130 print"prg file '";f$;"' created..."
GP  1140 print"this generator no longer needed."
CP  1150 :
AG  1000 data  76,  89,  20,  76,  64,  20,  76,  79
PB  1010 data  20,  76, 138,  20,  76,  45,  20, 216
LB  1020 data  32,  36, 192, 144,  15,  32, 248, 245
NB  1030 data 173,  13, 220, 173,   4,  10,  74, 144
JA  1040 data   3,  32,   6,  64, 173,   9,  21, 208
DH  1050 data  54, 172,   6,  21, 186, 138, 153, 250
BE  1060 data  20, 136, 192, 255, 208,   2, 160,   3
BJ  1070 data 185, 238,  20, 201,   0, 240, 242, 152
JI  1080 data 170, 254, 254,  20, 189, 254,  20, 221
NK  1090 data   2,  21, 208, 229, 169, 255, 157, 254
HD  1100 data  20, 140,   6,  21, 185, 246,  20, 141
ME  1110 data   9, 213, 185, 250,  20, 170, 154,  76
JM  1120 data  51, 255, 216, 169, 127, 141,  13, 221
EH  1130 data 172,  13, 221,  48,  20,  32,  61, 246
HM  1140 data  32, 225, 255, 208,  12,  32,  86, 224
HI  1150 data  32,   9, 225,  32,   0, 192, 108,   0
KE  1160 data  10,  32, 213, 232, 165, 211,  41,  15
AN  1170 data 240, 213, 162, 255, 232,  24,  74, 176
NE  1180 data   7, 224,   4, 208, 247,  76,  95,  19
MI  1190 data 189, 242,  20, 240,  51, 173,   7,  21
HP  1200 data 240,  19, 169,   0, 168, 153, 238,  20
GO  1210 data 200, 192,   4, 208, 248, 169,   1, 157
IO  1220 data 238,  20,  76, 208,  19, 189, 238,  20
KO  1230 data  73,   1, 157, 238,  20, 162,   4, 202
EO  1240 data 189, 238,  20, 201,   0, 208,   9, 224
HL  1250 data   0, 208, 244, 169,   1, 157, 238,  20
DB  1260 data 173,   8,  21, 240, 138, 173,   9,  21
KB  1270 data 208, 133, 169,  18,  32, 210, 255, 169
```

```
CD 1280 data 255, 133, 250, 230, 250, 166, 250, 224
KP 1290 data   4, 240,  30, 189, 242,  20, 240, 243
PJ 1300 data 138,  24, 105,  48,  72, 160, 155, 189
LD 1310 data 238,  20, 240,   2, 160,   5, 152,  32
EF 1320 data 210, 255, 104,  32, 210, 255,  76, 227
JI 1330 data  19, 169, 146,  32, 210, 255, 169,   5
DN 1340 data  32, 210, 255, 169,  13,  32, 210, 255
JB 1350 data  76,  51, 255, 234, 120, 174,   6,  21
PE 1360 data 169,   0, 157, 238,  20, 157, 242,  20
MI 1370 data  88, 234,  76,  41,  20, 176,  13, 153
PF 1380 data   2,  21, 185, 242,  20, 240,   4, 138
IK 1390 data 153, 238,  20,  96, 142,   9,  21,  96
CG 1400 data   8, 141,  13,  21, 142,  12,  21, 140
KE 1410 data  11,  21, 104, 141,  14,  21,  96, 141
AI 1420 data  10,  21, 142,  15,  21, 140,  16,  21
ID 1430 data  96, 120, 141,   8,  21, 142,   7,  21
GI 1440 data 162,  15, 160,  19, 142,  20,   3, 140
OP 1450 data  21,   3, 162,  98, 160,  19, 142,  24
AM 1460 data   3, 140,  25,   3, 162, 204, 160,  20
IN 1470 data 142,  48,   3, 140,  49,   3, 162, 222
KE 1480 data 160,  20, 142,  50,   3, 140,  51,   3
EB 1490 data  88,  96, 120, 170, 169,   0, 157, 238
FB 1500 data  20, 169,   1, 157, 242,  20, 169, 246
CE 1510 data 157, 250,  20, 169, 247, 133, 250, 189
GB 1520 data 246,  20, 133, 251, 173,  15,  21,  24
PP 1530 data 105, 255, 141,  15,  21, 173,  16,  21
IB 1540 data 105, 255, 141,  16,  21, 160,   0, 185
GN 1550 data  10,  21, 145, 250, 200, 192,   7, 208
BI 1560 data 246, 169,  27, 145, 250, 200, 169,  20
DE 1570 data 145, 250,  88,  96,  72, 169,   1, 141
PC 1580 data   9,  21, 104,  32, 108, 242,  72, 169
MK 1590 data   0, 141,   9,  21, 104,  96,  72, 169
HK 1600 data   1, 141,   9,  21, 104,  32,  78, 245
IF 1610 data 162,   0, 142,   9,  21,  96,   1,   0
OI 1620 data   0,   0,   1,   0,   0,   0,   1,  22
IF 1630 data  23,  24,   0,   0,   0,   0, 255, 255
MF 1640 data 255, 255,   0,   0,   0,   0,   0,   0
AE 1650 data   0,   0,   0,   0,   0,   0,   0,   0
CB 1660 data   0
```

Listing 4: PAL/Buddy-format source code listing for the multitasking system. When assembled, this creates the program "multi.ml".

```
IB  10 open 8,8,1,"0:multi.ml"
DC  20 rem open8,8,1,"0:multi.ls"
PO  30 sys 700
LE  40 .opt o8
EI  50 ; * * * * * * * * * * * * * * *
HH  60 ; *  multitasking for c128  *
JH  70 ; *           by            *
BH  80 ; *       mike mohilo       *
MK  90 ; * * * * * * * * * * * * * * *
KN 100 ;
DF 110 *=$1300
PJ 120 chrout = $ffd2
EO 130 kload  = $f26c          ;;kernal load/save that
EF 140 ksave  = $f54e          ;;bypass the jump table
HC 150 jmp init                ; a=disp x=otat
AI 160 jmp setregs             ;a=a    x=x    y=y    p=p
GK 170 jmp setprog             ;a=mmu  x=pcl  y=pch
AH 180 jmp create              ; a=task
CJ 190 jmp runstop             ;a=delay x=rnst y=task# c=pri
OD 200 ;
LC 210 irq cld                 ;;irq routine
OA 220 jsr $c024               ;;
EP 230 bcc swap                ;;
LI 240 jsr $f5f8               ;;
PL 250 lda $dc0d               ;;duplicate of
JO 260 lda $0a04               ;;kernal irq
IJ 270 lsr                     ;;
GC 280 bcc swap                ;;
AD 290 jsr $4006               ;;
EG 300 swap lda prirty         ;see if priority task
AH 310 bne rtnint
EJ 320 ldy ctask               ;get current task
LE 330 tsx
IA 340 txa
MG 350 sta stack,y             ;store stack pointer
EK 360 find dey                ;find a new task
BK 370 cpy #$ff
HE 380 bne gtask
FB 390 ldy #$03
FP 400 gtask lda runst,y       ;see if it is running
CA 410 cmp #$00
LN 420 beq find                ;not running-look again
FG 430 tya
DI 440 tax
NN 450 inc timer,x
HM 460 lda timer,x
CP 470 cmp delay,x
CO 480 bne find                ;task delayed-get another
HK 490 lda #$ff
GP 500 sta timer,x             ;reset timer
JE 510 sty ctask
KI 520 lda spage,y             ;get new stack page
MP 530 sta $d509
GP 540 lda stack,y             ;get new stack ptr
BP 550 tax
MC 560 txs
FN 570 rtnint jmp $ff33        ;kernal-return from interrupt
KL 580 ;
NF 590 nmi cld                 ;;nmi routine
JJ 600 lda #$7f                ;;
LK 610 sta $dd0d               ;;
HN 620 ldy $dd0d               ;;
OK 630 bmi next                ;;
ON 640 jsr $f63d               ;;
KA 650 jsr $ffe1               ;;duplicate of
FL 660 bne next                ;;kernal nmi
CO 670 jsr $e056               ;;
ON 680 jsr $e109               ;;
IN 690 jsr $c000               ;;
CH 700 jmp ($0a00)             ;
II 710 next jsr $e8d5          ;;
PO 720 lda $d3                 ;get shift/ctrl/cmdr/alt keys
NO 730 and #$0f
DB 740 beq rtnint              ;no keys
HA 750 ldx #$ff
FG 760 nextbit inx             ;convert key bits to #0-3
IH 770 clc
FO 780 lsr
AJ 790 bcs rstask
BM 800 cpx #$04
NN 810 bne nextbit
HI 820 jmp rtnint              ;no keys
IB 830 rstask lda crtbl,x      ;run/stop task
OK 840 beq display             ;task not created
JN 850 lda otat                ;one task at a time
JJ 860 beq togtask             ;not set-toggle task on/off
JI 870 lda #$00
PD 880 tay
DM 890 stopatk sta runst,y     ;stop all tasks
EG 900 iny
DD 910 cpy #$04
EF 920 bne stopatk
JM 930 lda #$01
PI 940 sta runst,x             ;run one task only
JF 950 jmp display
JI 960 togtask lda runst,x
IK 970 eor #$01                ;toggle task on/off
OK 980 sta runst,x
NG 990 ldx #$04
II 1000 deadlk dex             ;make sure at least 1 task runs
OI 1010 lda runst,x
EG 1020 cmp #$00
ID 1030 bne display            ;a task is running
BK 1040 cpx #$00
OK 1050 bne deadlk             ;look again
LE 1060 lda #$01
JM 1070 sta runst,x            ;all tasks stopped-run task #0
DB 1080 display lda dispt      ;display tasks
HG 1090 beq rtnint             ;no display
BN 1100 lda prirty
FA 1110 bne rtnint             ;do not disturb priority task
HE 1120 lda #$12               ;print a rvs-on
```

```
BE  1130 jsr chrout
BD  1140 lda #$ff
HJ  1150 sta $fa
EA  1160 dnext inc $fa          ;display tasks 0-3
JM  1170 ldx $fa
ND  1180 cpx #$04
HA  1190 beq dexit              ;no more tasks
HK  1200 lda crtbl,x            ;get a task
GG  1210 beq dnext              ;task not created
IH  1220 txa
EE  1230 clc
NF  1240 adc #$30               ;make #0-3 into ascii '0'..'3'
OF  1250 pha
PB  1260 ldy #$9b               ;task stopped-lt grey
CJ  1270 lda runst,x
MB  1280 beq color
DC  1290 ldy #$05               ;task running-white
OB  1300 color tya
PI  1310 jsr chrout             ;print color
AL  1320 pla
FP  1330 jsr chrout             ;print ascii task #
DM  1340 jmp dnext              ;look for another task
AI  1350 dexit lda #$92         ;print a rvs-off
HC  1360 jsr chrout
FH  1370 lda #$05               ;make color white
LD  1380 jsr chrout
PG  1390 lda #$0d               ;print a cr
PE  1400 jsr chrout
PN  1410 jmp $ff33              ;kernal rti
CA  1420 ;
GH  1430 kill nop               ;kill task
JD  1440 sei
CF  1450 ldx ctask              ;what task is this
HN  1460 lda #$00
GD  1470 sta runst,x            ;stop it
BL  1480 sta crtbl,x            ;un-create it
AG  1490 cli
IN  1500 idle nop               ;task will die after next irq
KG  1510 jmp idle
GG  1520 ;
HJ  1530 runstop bcs priority ;run/stop/delay task
IC  1540 sta delay,y            ;set delay timer
AB  1550 lda crtbl,y
HL  1560 beq notask             ;task not created
GN  1570 txa
OO  1580 sta runst,y            ;run/stop the task
MM  1590 notask rts
FJ  1600 priority stx prirty    ;set priority
GD  1610 rts
KM  1620 ;
HF  1630 setregs php            ;set a,x,y,p registers
DM  1640 sta rega
FI  1650 stx regx
HJ  1660 sty regy
OA  1670 pla
HC  1680 sta regp
GI  1690 rts
KB  1700 ;
OB  1710 setprog sta regm       ;set bank,starting address
FK  1720 stx rpcl
DK  1730 sty rpch
IL  1740 rts
ME  1750 ;
FB  1760 init sei               ;initialize program
IN  1770 sta dispt              ;display task option
BO  1780 stx otat               ;one task at time option
PL  1790 ldx #<irq
JM  1800 ldy #>irq
JC  1810 stx $0314
GG  1820 sty $0315              ;set irq vector
KN  1830 ldx #<nmi
EO  1840 ldy #>nmi
FF  1850 stx $0318
NG  1860 sty $0319              ;set nmi vector
DO  1870 ldx #<tload
NO  1880 ldy #>tload
NH  1890 stx $0330
AC  1900 sty $0331              ;set load vector
PA  1910 ldx #<tsave
JB  1920 ldy #>tsave
HK  1930 stx $0332
ME  1940 sty $0333              ;set save vector
MC  1950 cli
EJ  1960 rts
AB  1970 create sei             ;create task
HI  1980 tax
JO  1990 lda #$00
GO  2000 sta runst,x            ;don't run it yet
BA  2010 lda #$01
EH  2020 sta crtbl,x            ;make it 'created'
LG  2030 lda #$f6
DC  2040 sta stack,x            ;set the stack ptr
DI  2050 lda #$f7
FC  2060 sta $fa
PO  2070 lda spage,x            ;get the stack page
MD  2080 sta $fb
PP  2090 lda rpcl               ;-adjust program start address
DN  2100 clc                    ;-net effect is addr=addr-1
OH  2110 adc #$ff               ;-
KG  2120 sta rpcl               ;-
GC  2130 lda rpch               ;-
MJ  2140 adc #$ff               ;-
IH  2150 sta rpch               ;-
DP  2160 ldy #$00
FD  2170 initsk lda ntreg,y     ;build a stack
HF  2180 sta ($fa),y            ;put mmu,y,x,a,p,pcl,pch
FP  2190 iny                    ; registers on the stack
JE  2200 cpy #$07
NB  2210 bne initsk
AM  2220 lda #<kill             ;put a return address to
PI  2230 sta ($fa),y            ;kill-routine on stack
LN  2240 iny ;
CN  2250 lda #>kill             ;when task ends with rts it
EP  2260 sta ($fa),y            ;will return to kill
MG  2270 cli
EN  2280 rts
IG  2290 ;
CP  2300 tload pha              ;trapped load
NC  2310 lda #$01
DJ  2320 sta prirty             ;get priority
CK  2330 pla
KG  2340 jsr kload
KK  2350 pha
LF  2360 lda #$00
FA  2370 sta prirty
EN  2380 pla
CE  2390 rts
GN  2400 ;
GG  2410 tsave pha              ;trapped save
LJ  2420 lda #$01
BA  2430 sta prirty             ;get priority
AB  2440 pla
KN  2450 jsr ksave
LB  2460 ldx #$00
FM  2470 stx prirty
MJ  2480 rts
AD  2490 ;
KP  2500 runst  .byte $01,$00,$00,$00 ;run/stop status
PO  2510 crtbl  .byte $01,$00,$00,$00 ;created task table
EH  2520 spage  .byte $01,$16,$17,$18 ;stack page table
CC  2530 stack  .byte $00,$00,$00,$00 ;stack pointer table
AK  2540 timer  .byte $ff,$ff,$ff,$ff ;delay timers
NC  2550 delay  .byte $00,$00,$00,$00 ;delay values
AJ  2560 ctask  .byte $00             ;current task
KJ  2570 otat   .byte $00             ;one task at time option
FD  2580 dispt  .byte $00             ;display tasks option
HP  2590 prirty .byte $00             ;priority task flag
HB  2600 ntreg = * ;new task registers
GH  2610 regm   .byte $00             ;mmu $ff00
FG  2620 regy   .byte $00             ;y
LG  2630 regx   .byte $00             ;x
JB  2640 rega   .byte $00             ;a
PF  2650 regp   .byte $00             ;p
OM  2660 rpcl   .byte $00             ;pcl
AN  2670 rpch   .byte $00             ;pch
OO  2680 ;
```

Exploring SUBMIT

Notes from the CP/M Plus workbench

by Adam Herst
Copyright (c) 1988 Adam Herst

SUBMIT is one of the most useful tools provided with CP/M Plus. It allows you to automate many of the repetitive tasks that are performed on a regular basis. Almost any series of commands that can be entered through the command line and executed by the CCP can be executed through a SUBMIT file.

The documentation provided with CP/M Plus covers the basic operation of SUBMIT. As is often the case, the documentation raises more questions than it answers. Unfortunately, information about the version of SUBMIT provided with CP/M 2.2 is not applicable. While their function is the same, their underlying processes are different. When the documentation fails, there is only place to go to get accurate information - your computer!

When a submit file is executed, SUBMIT rewrites the original file to a temporary file. You can verify this with the submit file, TEST01.SUB:

```
dir
```

Execute the file with the command:

```
SUBMIT TEST01.SUB
```

(Typing 'SUBMIT' on the command line is unnecessary. Submit files can be executed as if they were command files by setting your CP/M environment with the command:

```
setdef [order=(com,sub)]
```

The command 'SUBMIT' will be omitted from examples in the remainder of this article.)

The directory of the current user area on the current disk will be listed to the screen. In it should be a file with a filetype of $$$, the standard filetype for a CP/M system temporary file. The complete file specification is SYSIN56.$$$. The significance and origin of the number in the file specification remain a mystery to me.

When submit files are nested (a submit file is called from within another submit file), the numbers generated for the file specification follow a pattern. Submit file TEST02.SUB illustrates:

```
dir
submit test02
;
```

Note the semicolon (;) on the last line. It is required so that a new file specification will be generated. If it (or any other additional line(s)) is not included, SUBMIT will delete the original temporary file (because the last command line has been reached) before creating the temporary file for the nested submit file. Consequently, the SYSIN56.$$$ file specification will be reused ad infinitum.

Execute the submit file with the command:

```
TEST02
```

As the nested submit files are executed, directory listings are printed to the screen. Subsequent listings contain an additional temporary file entry. The numbers in the file specifications begin at 56 and decrease by one, skipping numbers ending in 8 and 3. (The submit file can be aborted with a CTRL-C when you grow tired of watching the screen.) ' "Curiouser and curiouser" said Alice', and I'm inclined to agree.

How deeply can submit files be nested? One guess would be to 46 levels. This would be the limit placed on the generation of temporary file names, if the numbers in the file specification stopped at 0 (a logical assumption). TEST02.SUB can be used to check this. This time, instead of stopping the submit file, let it run it's course.

The number in the temporary file specifications never reaches 0. When the number in the file specification has reached 16, the next nested submit file causes the following error message to be displayed:

```
CANNOT LOAD PROGRAM
```

to be displayed. Apparently, submit files can only be nested to a maximum of 33 levels on the C-128.

This limit is imposed by available memory and is a result of the operation of SUBMIT under CP/M Plus. SUBMIT uses the 'Resident System Extension' (RSX) capabilities of CP/M Plus. As the name implies, RSX's can be attached to the standard operating system to handle custom tasks as system functions. Each time SUBMIT is invoked, it attaches an RSX to process the submit file. To attach an RSX, sufficient memory at the top of the 'Transient Program Area' (TPA) must be available. When an RSX is attached, the amount of available high memory is reduced. When SUBMIT has nested 33 levels, it appears that either high memory has dropped so low that there is no room left for SUBMIT to be loaded and executed or that there is no room for SUBMIT to attach its RSX. This also implies that the presence of other RSX's attached to the system will reduce the number of levels that submit files can be nested.

The creation of a temporary file holds a number of implications for the use of SUBMIT. First, there must be enough space on the disk to hold the temporary file. Also, the creation of a temporary file is a factor in the execution time of SUBMIT files.

How big can a submit file be? Since a temporary file is created from it, it must be smaller than the remaining space on the temporary file disk. If it isn't, SUBMIT will abort execution and print the message:

DISK WRITE ERROR: LINE nnnn

where nnnn is the line SUBMIT was trying to write to the temporary file when it ran out of space.

As long as there is space on the disk for the temporary file, there is no apparent limit on the size of a submit file. I have successfully created a submit file the full size of the RAM disk, and was able to execute it by having the temporary file written to the 1581. That's a 512K submit file, the largest that I can test on my system, and larger than I have ever actually needed.

(The drive that is to be used for the storage of temporary files can be designated with the command:

SET [TEMPORARY=d:]

where d: is the drive specification of the temporary drive.)

The creation of a temporary file is a factor in the execution time of SUBMIT files. Disk access is the processing bottleneck on the C128; consequently, SUBMIT file execution times should be influenced by strategic selection of the drive on which the temporary file will be written. Two considerations are relevant in choosing the location of the temporary file: the speed of the temporary drive, and the location of the submit file.

It seems obvious that SUBMIT execution times will decrease with increases in the speed of the temporary drive - the temporary file will be created faster, and the command lines will be retrieved from it faster. The influence of the location of the submit file, and the effect of its interaction with drive speed, is not as clear cut. Consider this - if the temporary file is to be created on the same drive as the submit file, the drive will be forced to access two separate locations on the same disk, imposing overhead in the form of additional drive head movement, and slowing down submit file execution.

To examine the effects of the various combinations of submit and temporary file locations, the following submit files, TIME.SUB and DUMMY.SUB, can be used. They are well commented and shouldn't require additional explanation. Well, maybe 'comments' require explanation.

Comments are a feature of the CCP rather than of SUBMIT. Any line beginning with a semicolon (;) is echoed to the screen and not interpreted or executed by the CCP. Since SUBMIT simulates a user entering commands at the keyboard, comment lines in a SUBMIT file are echoed to the screen and not executed.

Create the file TIME.SUB:

```
; time.sub 1/21/88
;
; time the effects of changing the locations
; of the submit and temporary files
; $1 is the location of the submit file
; $2 is the location of the temporary file
;
; set the temporary drive
setdef [temporary=$2]
;
; set the starting time to 0
m:conf date = 00:00:00
;
; submit the file to be timed
m:submit $1dummy.sub
;
; show the execution time
m:conf date
;
; reset temporary drive - PLACE YOUR PREFERENCE HERE
setdef [temporary=m:]
```

Create the file DUMMY.SUB:

```
; dummy.sub 1/25/88
; This is a dummy submit file. It causes a temporary
;  file to be written to the selected disk.
```

To determine submit file execution time, place both submit files on a disk in each drive that is to take part in the test. Also make sure that the necessary support files (e.g. CONF, SUBMIT) are located on the designated drives in accessible user areas. Invoke the test with the command:

SUBMIT TIME <submit file drive> <temporary file drive>

where 'submit file drive' is the drive from which DUMMY.SUB will be loaded and 'temporary file drive' is the drive to which

SUBMIT is to write its temporary file. Don't forget to provide both parameters on the command line! The time required to read the submit file, write the temporary file, and then echo the commands from the temporary file to the screen will be shown at the bottom of the screen as the current date/time.

Here are the times generated on my machine:

Submit File Drive	Temporary File Drive		
	1571	1581	1750
1571	13	10	6
1581	14	9	5
1750	12	6	5

All times are in seconds. Note that these are not benchmarks. You are left to draw your own conclusions as to their final meaning and significance for your system setup. (My temporary file drive is set to the 1750 RAM disk. The location of the submit file does not seem to be significant in that case.)

How does SUBMIT use the temporary file? One of its purposes is to allow the substitution of command line parameters to be performed. Parameters are represented in submit files by a dollar sign followed by a digit from one to nine ($1 - $9). When a submit file is invoked, any arguments following the name of the submit file are substituted for the appropriate parameter in the submit file. Submit file TEST03.SUB makes use of comment lines as well as illustrating parameter passing:

```
;
;
;   $1 $2 $3 $4 $5 $6 $7 $8 $9 $10
;
;
type sysin56.$$$
```

Execute TEST03.SUB with a command line of:

```
TEST03 WHAT IS THE NAME OF THE SUBMIT FILE BEING EXECUTED
```

The temporary file will be printed to the screen. The reference to $1 in the submit file has been replaced with 'WHAT', reference to $2 with 'IS' and so on. The reference to parameter $10 has become 'WHAT0', not the 'EXECUTED' you might have expected. Only the first digit after the dollar sign is considered significant for parameter replacement.

(When SUBMIT encounters a dollar sign that is not followed by a digit, in all but one case, the error message:

```
PARAMETER ERROR IN LINE nnn
```

will be printed and execution will be aborted. If SUBMIT encounters a dollar sign followed by a dollar sign however, it will replace the dollar sign pair with a single dollar sign. In this way, a dollar sign may be included in a submit file for some purpose other than parameter passing.)

One digit is missing in this analysis - $0. This is documented as a valid parameter in the DRI manual set, but no mention is made of its function. We can find out by changing the line with the parameters in TEST03.SUB to read:

```
;  $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
```

and removing the line:

```
type sysin56.$$$
```

Execute the modified TEST03.SUB with the command line:

```
TEST03 IS THE NAME OF THE SUBMIT FILE BEING EXECUTED
```

The comment line containing the parameters is echoed to the screen with the appropriate substitutions. $0 has been replaced with the name of the submit file.

Another feature of the CCP that takes on special significance in submit files is 'Conditional Command Execution'. Programs that run under CP/M Plus can set a 'Program Return Code'. The CCP initializes this code to *successful* before a program is run. If the program encounters an error condition, it can set the return code to *unsuccessful* before it terminates. Additionally, the CCP will set the return code to *unsuccessful* if the program terminates with a fatal BDOS error or a CTRL-C. Command lines in a submit file that begin with a colon will not be executed if the previously executed command has set the program return code to *unsuccessful*. Submit file TEST05.SUB illustrates:

```
$1 $2 $3 $4 $5 $6 $7 $8 $9

: dir
```

Execute TEST05.SUB with various arguments designed to force some kind of error. If the error code is set to *unsuccessful*, the command on the line containing the colon will not be executed. Try anything you can think of - the conditional command line will always execute. I have not found a single program or utility, let alone a CP/M Plus command, that will set the return code to *unsuccessful* as the result of a program error.

Just to make sure that conditional command execution does work, execute TEST05.SUB with a command line of:

```
TEST05 DIR J:
```

Attempting to access drive J: (a non-existent drive on most C-128 CP/M systems) causes a fatal BDOS error. Consequently, the CCP sets the program return code to *unsuccessful* and the conditional command line is not executed.

I'll close this look at SUBMIT with some submit files that I use often. They are simple examples that show how SUBMIT can be used to perform a variety of repetitive tasks. They also illustrate how characteristics of other CP/M commands can be used to extend the capabilities of submit files.

MESSAGE

There is no formal way to cause a submit file to pause during its run, short of terminating it. This can be a problem when a submit file is to perform operations involving more than one disk. No opportunity is given to insert the appropriate disk in the appropriate drive. Also, there is no way to indicate that a disk switch is necessary or which disk(s) are to be switched.

MESSAGE.SUB offers a solution to both problems. It echoes a message of up to nine words to the screen, then pauses the submit file until a signal from the user is received. Parameters and comment lines are used to echo the message. A characteristic of PIP is used to pause execution of the submit file.

PIP can copy to and from devices as well as files. One device under CP/M is CON:, representing the keyboard in an input context and the screen in an output context. When PIP copies from CON:, it waits for a predetermined signal (CTRL-Z, the end-of-file character) to end the process. This effectively halts submit file execution until the PIP operation is finished.

```
;
;
;
;
;
;
;
;
;
;
;    $1 $2 $3 $4 $5 $6 $7 $8 $9
;
;
;
;
;
;
;
;
;
;
;
;    Press CONTROL-Z to continue...
;
pip con:=con:
```

Execute MESSAGE.SUB with the command:

```
MESSAGE <your message>
```

where <your message> can contain up to nine words delimited with spaces. The message is optional - omitting it causes only the 'Press CONTROL-Z to continue...' message to be displayed.

STRIP and SPACE

STRIP and SPACE perform two very common operations on text files. STRIP will strip excess carriage return/linefeed combinations from a named file. SPACE will double the line spacing of a named file. The engine driving these two submit files is the DRI context editor, ED. You may want to review the command syntax of ED in the DRI documentation to modify these submit files to meet your own needs.

STRIP treats as excess any carriage return/linefeed that is not followed by another carriage return/linefeed (i.e it ends a line rather than a paragraph). STRIP removes the excess characters and joins the lines that had been separated. If a text file consists *only* of excess carriage return/linefeeds, the resultant file will consist of a single line of text. Create the file STRIP.SUB:

```
; strip.sub 1/15/88
;
era stripped.txt
pip stripped.txt=$1
ed stripped.txt
<mn^L^L^Z-4c#s^L^L^Z**^Z
<h
<mn^L^Z-2c#s^L^Z ^Z
<h
<mn**^Z-2c#s**^Z^L^L^Z
<h
<e
era stripped.bak
```

Execute STRIP with the command line:

```
STRIP <filename>
```

where filename is the name of the file to be stripped. STRIP writes the stripped file to the file STRIPPED.TXT - the original file remains unchanged. If the marker characters '**' are used in your text file, you will want to change the marker characters in the submit file to some unused character string.

SPACE doubles the line spacing of the named file. It replaces every single carriage return/linefeed with two carriage return/linefeeds. If your text was single-spaced, SPACE will make it double-spaced. If it was double-spaced, SPACE will make it quadruple-spaced. Create the file SPACE.SUB:

```
; space.sub 1/15/88
era spaced.txt
pip spaced.txt=$1
ed spaced.txt
<mn^L^Z-2c#s^L^Z^L$2^Z
<h
<e
era spaced.bak
```

Execute SPACE with the command line:

```
SPACE <filename>
```

where filename is the name of the file to be spaced. SPACE writes the spaced file to the file SPACED.TXT - the original remains unchanged.

A Machine Language Input Routine

A routine for all reasons

By Garry Kiziak

I'm sure that many of you, like myself, get a great deal of pleasure out of writing your own programs - even when there is a commercial program available that will accomplish the same thing. It's the pride and the sense of accomplishment that we get when we complete that last line and say "There! It's done, and it works!" In many cases, the result is even better than the commercial program, if only because it was designed specifically to meet your needs and not somebody else's.

Often, the one thing that distinguishes a commercial quality program from one that you create yourself is the manner in which input is obtained from the user. Let's face it, the INPUT statement in Commodore BASIC is not the most useful command to use. Here are some of its limitations - I'm sure you can think of more:

1. If you enter a comma or a colon, an 'extra ignored' error message is displayed.

2. If you only want numeric data to be entered and the user enters an alphabetic character, you will get a '?redo from start' error.

3. You can enter control characters in the middle of the input. (e.g. Press the CLR/HOME key in the middle of an input and watch the screen clear. Similar problems arise from the cursor keys, the delete key, and others.)

4. The user can type as many characters as he likes, often destroying the appearance of the screen that you spent many hours designing.

5. Often you would like the user to be able to type only certain characters (e.g. 'y' or 'n'), yet he can type any character that he wants.

We have learned how to get around many of these problems. For example, to get around problem 2 above, simply use a string variable to get the input and then the VAL command to convert it to a numeric. Of course, a certain amount of error trapping has to be done along with this to make sure that the user doesn't enter 0 accidentally (i.e. by entering an alphabetic character instead). Similarly, problem 5 could be eliminated by some error trapping.

'Bullet-proof' input subroutines

The most common way around this problem is to not use the INPUT command at all. Instead a special 'bullet-proof input subroutine' is written and used whenever an INPUT command would normally be used. Many such routines have appeared in magazines, user club libraries, etc. I'm sure you have your own favourites and have probably created some yourself. There are some problems with this approach, however.

1. Most such routines make use of the GET statement. As a consequence a premature garbage collection can occur, resulting in an annoying delay in your program.

2. Usually these routines are very specific (e.g. one may only allow alphabetic input, another may only allow numeric input, etc.). If this is the case, you may need several such routines in your program (one for each type of input that you require). This can eat up a lot of memory fast.

3. If the routine is, in fact, versatile (that is, it is able to handle many different types of input), it will likely be slow and probably consume a lot of memory. I wrote such a routine once when I was creating a database program. The routine eventually took up over 1K of memory but even more important it was terribly slow. A reasonably fast typist would either lose some of the characters typed or else have to slow down their typing speed to adjust to the program.

The program in this article will offer you an alternative. It is another 'bullet-proof input routine', but written in machine language instead. It still is fairly long (854 bytes to be exact) but it is stored in a place in memory that won't take away from your BASIC programs. Because it is written in machine language, you won't have to slow down when typing in your

input. It does not force any premature garbage collection, and is quite versatile - in fact, as you will see, it allows you to do a lot of the things that you normally do when editing your BASIC programs.

First, type in the assembly language program (Listing 1). Don't forget to save it in case you make a mistake or you want to modify it later (some suggestions are given at the end of the article). Assemble it and save the resulting machine language as "input.obj". If you don't have an assembler, type in Listing 3. This will create the input.obj file on your disk automatically. Then type in the BASIC program (Listing 2), which is a short demonstration of the capabilities of this routine.

Using the subroutine

Before I explain what the demo does, perhaps I should explain the syntax of the calling statement in BASIC, which is.

sys in,x,y,in$,le,id[,b$]

The square brackets indicate that the ",b$" is optional.

In this statement:

in is the calling address of the machine language routine. If you assemble it where I have, in=49155 (see line 30 in the BASIC demo).

x is the column that you want the input to begin in (0-39).

y is the row that you want the input to be on (0-24).

**in$** is the string variable which will receive the input. You must initialize in$ prior to using this routine; either to a string of blanks, or to whatever you would consider to be the default input. This string is printed to the screen when the input routine is entered. If it is blank, everything appears as in a normal input statement except for the question mark. If you provide a default input, it can be edited using the cursor keys, insert and delete keys, etc., just as you would edit a BASIC program - the difference being that not all keys are active, just those that you specify.

le is the length of the input; that is, the maximum number of characters that you want to allow the user to enter. This number must be less than or equal to the length of the string in$, otherwise you will get an 'illegal quantity' error message. The entire string in$ is, in fact, not printed as stated above, just the first 'le' characters.

id is the identification number. This is what determines what keys are active on input and what features are in force.

Selective input

When id=1, only alphabetic characters, upper and lower case, and a space are allowed. Everything else is ignored.

YOU CAN HAVE IT ALL
THE CONVENIENCE OF A CARTRIDGE!
THE FLEXIBILITY OF A DISK!

THE QUICK BROWN BOX stores up to 30 of your favorite programs - Basic & M/L, Games & Utilities, Word Processors & Terminals - READY TO RUN AT THE TOUCH OF A KEY - HUNDREDS OF TIMES FASTER THAN DISK - Modify the contents instantly. Replace obsolete programs, not your cartridge. Use as a permanent RAM DISK, a protected work area, an autoboot utility. C-64 or C-128 mode. Loader Utilities included. Price: 16K $69 32K $99 64K $129 (Plus $3 S/H; MA res add 5%) 30 Day Money Back Guarantee. 1 Year Warranty. Brown Boxes, Inc, 26 Concord Road, Bedford, MA 01730; (617) 275-0090

THE QUICK BROWN BOX - BATTERY BACKED RAM
THE ONLY CARTRIDGE YOU'LL EVER NEED

Free Spirit Software Inc.

"... excellent, efficient program that can help you save both money and downtime." — *Compute!'s Gazette Dec. 1987*

1541/1571 DRIVE ALIGNMENT

1541/1571 Drive Alignment reports the alignment condition of the disk drive as you perform adjustments. On screen help is available while the program is running. Includes features for speed adjustment. Complete instruction manual on aligning both 1541 and 1571 drives. Even includes instructions on how to load alignment program when nothing else will load! Works on the C64, SX64, C128 in either 64 or 128 mode, 1541, 1571 in either 1541 or 1571 mode! Autoboots to all modes. Second drive fully supported. Program disk, calibration disk and instruction manual only **$34.95!**

SUPER 81 UTILITIES

Super 81 Utilities is a complete utilities package for the 1581 disk drive and C128 computer. Among the many Super 81 Utilities features are:
- Copy whole disks from 1541 or 1571 format to 1581 partitions.
- Copy 1541 or 1571 files to 1581 disks
- Backup 1581 disks or files with 1 or 2 1581's
- Supplied on both 3½" and 5¼" diskettes so that it will load on either the 1571 or 1581 drive.
- Perform many CP/M and MS-DOS utility functions.
- Perform numerous DOS functions such as rename a disk, rename a file, scratch or unscratch files, lock or unlock files, create auto-boot and much more!

Super 81 Utilities uses an option window to display all choices available at any given time. A full featured disk utilities system for the 1581 for only **$39.95!**

RAMDOS RAM-DISK

RAMDOS is a complete RAM based "Disk" Operating System for the Commodore 1700 and 1750 RAM expansion modules which turns all or part of the expansion memory into a lightning fast RAM-DISK. RAMDOS behaves similar to a much faster 1541 or 1571 floppy disk except that the data is held in expansion RAM and not on disk. Under RAMDOS, a 50K program can loaded in ½ second. Programs and files can be transferred to and from disk with a single command. RAMDOS is available for only **$39.95!**

Order with check, money order, VISA, MasterCard, COD. Free shipping & handling on US, Canadian, APO/FPO orders. COD & Foreign orders add $4.00

Order From **Free Spirit Software, Inc.**
905 W. Hillgrove, Suite 6
LaGrange, IL 60525
(312) 352-7323
1-800-552-6777
For Technical Assistance call: (312)352-7335

When id=2, only the numeric characters 0-9 are allowed. Use this id to accept integers as input. See below for floating point numbers.

The identification number is additive in the sense that if id=3 (i.e. 1+2), both the alphabetic and numeric characters are allowed. Everything else is ignored.

When id=4, the period is allowed as a decimal point. This would be used along with 2 (i.e. id=6) to allow the input of decimal numbers. Because this is a decimal point, it can be entered only once in a given input. Of course, it could be deleted and then entered elsewhere in the same input, if the user so desires.

When id=8, cursor up and cursor down keys act just like the return key (i.e. they terminate the input). You can tell which key terminated the input by peeking at location 780. If it contains a 1, the return key was pressed. If it contains a 2, the cursor down key was pressed, and if it contains a 3, the cursor up key terminated the input.

When id=16, function one key (F1) can be used as an escape key. Input is of course terminated as if you pressed the return key (or the cursor up/down key), but you can tell if the F1 key was pressed by peeking at location 781. If it contains a 0, the F1 key was not pressed. If it contains a 1, the F1 key was pressed.

When id=32, any trailing blanks are removed from in$.

When id=64, the default input is left justified when the input is entered but right justified when the input routine is exited (see the BASIC demo for an example).

As you may have noticed, no provision has been made for allowing characters such as the dollar sign, the comma, etc. An id of 128 overcomes this. When id=128, the ',b$' must be included in the calling statement and any characters stored in the variable b$ will be allowed to be entered (e.g. if b$="+-/*" and id=128, the four arithmetic operators may be entered into the input).

As stated above, the identification number is additive. Thus if id=51 (i.e. 1+2+16+32), only the upper and lower case alphabetic characters and the numeric digits 0-9 will be allowed. Furthermore, the F1 key can be used as an escape key and any trailing blanks that remain in the input variable will be removed. Also notice that a space can always be input regardless of the identification number.

The BASIC demo

A brief explanation of the BASIC demo is now in order. Notice first that there is another ML routine included with the input routine. It is a 'print at' routine. The command sys pr,x,y,a$ (pr is initialized in line 130 as well) will print the contents of a$ at location x,y of the screen.

Line 100 simply loads the machine language routines into memory.

Lines 130-140 initialize various variables - pr and in as indicated before. The variable 'ret' is the location to be peeked to determine if the input was terminated using the return key or the cursor keys, while 'esc' is the location to be peeked to determine if the escape key (i.e. F1) was pressed. b$, c$, and d$ are used below.

Lines 150-220 print a blank template on the screen for what could be a database program.

Lines 230-260 initialize several variables with data that will be placed in this template to be modified by the user.

Lines 1000-1150 allow the user to modify the data using the input routine. For example, the input command in line 1000 allows the user to modify the name. An identification number of 153 is used (i.e. 1+8+16+128). Thus only the upper and lower case alphabetic characters and the character in b$ (i.e. the period) can be used. The cursor up/down keys can be used to terminate input and the F1 key can be used as an escape key. The third statement in line 1000 checks to see if the escape key was pressed. If it was, control passes to line 1160 which quits the program. If it wasn't, line 1010 checks to see what key terminated the input. If it was the return key or the cursor down key, control is passed to line 1020. If it was the cursor up key, control is passed to line 1140.

The remaining lines behave similarly.

Run the program and notice how the cursor left/right keys work during an input. Also notice how the delete/insert keys work - they should be identical to the way they work when editing a BASIC program. Depending on what field you are editing, only certain keys are permitted, the others are ignored (see if you can predict which ones are permitted by looking at the id). The cursor down key (or the return key) moves you to the next field of data and the cursor up key moves you to the previous field. Wraparound is in effect in both cases. You can edit any field and move from field to field as often as you like. Also notice how quickly you can move from field to field (simply hold down the cursor down or cursor up key).

When you get to the 'Amount owed:' field, notice how the input jumps to the left when you begin to enter something and jumps back to the right when you exit the field.

To quit the program, simply press the escape key in any field.

Some comments and suggestions

If you analyze the assembly language routines, you will notice that whatever the user types is stored directly into memory exactly where the original data for the variable in$ is stored. It does not create a new string. Consequently, a premature garbage collection will not result from the use of this routine.

It also has a side effect. If you didn't make any changes to the data when you ran the program initially, run it again and change the name or the address or whatever. Then quit the program and list it. You will see that the data statements in lines 240-260 will have changed accordingly. This problem should occur only rarely because the variables that you create will normally be stored in high memory, not within the BASIC program itself. To eliminate this problem, all you have to do is force your variables to be stored in high memory - a statement such as na$=na$+"" will do this.

It would be an interesting exercise to modify this program to better suit your own needs. For example, frequently when a default input is presented, it is not acceptable to the user. At present, the user must type over the default and erase anything that is left over. Modify the program so that pressing the CLR/HOME key will blank out the default input.

I'm sure that in your programming experiences, you have encountered many other types of input restrictions that would be useful in a program. Modify the program to incorporate these. Some suggestions are:

1. Convert all lower case characters to capitals as they are entered. This would be useful when designing educational programs for use by elementary students.

2. Don't allow a space as the first character in an input or else remove any leading spaces that are input without changing the length of the string (i.e. left-justify the input).

3. Terminate the input on entering the last character in the input field.

4. Skip over certain characters (e.g. skip over the /'s in the date 12/24/87 or skip over the -'s in the phone number 999-999-9999, etc.).

5. Convert the first character after a space to a capital. This is for the lazy typist who doesn't want to use the shift key when typing in names.

You can either remove features already in the routine and replace them, or you can add new ones to those already in place. If you choose the latter, note that the variable ID in the assembly language routine which is presently an eight bit 'mask' would have to become a sixteen bit or bigger mask. This complicates things a little, but the challenge should spur you on.

Some other modifications to consider are to change the flashing cursor into a solid cursor or into an underline cursor, or you may simply want to change the rate at which the cursor flashes.

I hope you find this a useful routine as is. I certainly have. If you do make any modifications, don't hesitate to send me a copy. I'm always interested in seeing what other people can do, especially when I have given them a starting point.

Listing 1: "input.src" (PAL format)

```
JL 1000 sys 700
KF 1010 .opt oo
CH 1020 ;
EH 1030 ; ************************
KD 1040 ; *                      *
JN 1050 ; *  m/l input routine    *
OL 1060 ; *   copyright 1987     *
CD 1070 ; *   garry g. kiziak    *
CG 1080 ; *                      *
AL 1090 ; ************************
CM 1100 ;
CA 1110 *=$c000 ; origin of routines
GN 1120 ;
NO 1130 ; command jump table
KO 1140 ;
ED 1150 jmp print ; print at routine
GA 1160 jmp input ; input routine
IA 1170 ;
PK 1180 ; get cursor position
MB 1190 ;
BG 1200 chkcom = $aefd ; check for a comma
NP 1210 combyt = $b7f1 ; get a byte in x
HB 1220 illqty = $b248 ; illegal quantity
EJ 1230 plot = $fff0 ; set/read cursor position
EJ 1240 xval .byte 0 ; temporary storage
AK 1250 yval .byte 0 ; temporary storage
CG 1260 ;
EI 1270 getcur jsr combyt ; get column
MA 1280 cpx #$28 ; 0<=x<=39
PI 1290 bcs set1 ; too big
JP 1300 stx yval
CN 1310 txa
EK 1320 pha
CA 1330 jsr combyt ; get row
BE 1340 cpx #$19 ; 0<=y<=24
LM 1350 bcs set1 ; too big
ED 1360 stx xval
CO 1370 pla
DD 1380 tay
EO 1390 clc
HD 1400 jmp plot ; set cursor
CE 1410 set1 jmp illqty
CA 1420 ;
IP 1430 ; print at routine
GB 1440 ;
JA 1450 print jsr getcur
CD 1460 jsr chkcom
KG 1470 jmp $aaa4 ; continue with rom print
OD 1480 ;
OA 1490 ; wait for a keystroke
CF 1500 ;
EE 1510 getin = $ffe4 ; check for a keypress
FI 1520 beg = $fb ; beginning of input field
DJ 1530 curpos = $fd ; cursor position within input field
KH 1540 ;
BO 1550 getkey lda ir ; get character under cursor
PH 1560 eor #$80 ; reverse it
FL 1570 sta ir
BF 1580 ldy curpos ; get cursor position
PP 1590 sta (beg),y
OA 1600 lda #$10 ; initialize counter
AM 1610 sta count2
BB 1620 lda #$ff
CN 1630 sta count1
NF 1640 get1 jsr getin ; has a key been pressed
JP 1650 bne get2 ; yes
```

```
ML 1660 dec count1 ; count down
MK 1670 bne get1 ; try again
KD 1680 dec count2 ; count down some more
AM 1690 bne get1 ; try again
MF 1700 beq getkey ; flash cursor
IM 1710 get2 rts
KK 1720 count1 .byte 0 ; counter for flashing cursor
BA 1730 count2 .byte 0
CE 1740 ;
EB 1750 ; input routine
GF 1760 ;
JN 1770 len = $02 ; max. no. of characters allowed
HN 1780 ast = $03 ; address of input string
OD 1790 lenb = $b2 ; length of optional string
AL 1800 bst = $b3 ; address of optional string
PF 1810 varadr = $05 ; address of variable
BD 1820 findvar = $b08b ; find variable
BF 1830 justf .byte 0 ; justify flag
FH 1840 escflg .byte 0 ; escape flag
AB 1850 iq .byte 0 ; character being entered
PL 1860 ir .byte 0 ; character under cursor
OF 1870 id .byte 0 ; mask for allowable inpputs
CM 1880 ;
DE 1890 input lda #$00
MK 1900 sta justf ; no justfication
PE 1910 jsr getcur ; get cursor position
GP 1920 clc
BA 1930 lda $d1 ; get screen address
IG 1940 adc $d3 ; for beginning of input
FO 1950 sta beg
CF 1960 lda $d2
HM 1970 adc #$00
MN 1980 sta beg+1
EE 1990 jsr chkcom
FE 2000 jsr findvar ; find input variable
IK 2010 sta varadr ; save its location
GG 2020 sty varadr+1
LD 2030 ldy #$02 ; move its descriptor
PH 2040 inp1 lda (varadr),y ; to zero page
OM 2050 sta len,y
HM 2060 dey
DF 2070 bpl inp1
LE 2080 lda len
OF 2090 beq set1
FE 2100 jsr combyt ; get max length of input
CP 2110 txa
MH 2120 beq set1
BJ 2130 cpx len ; bigger than length of string .
NM 2140 beq inp1a
JJ 2150 bcc inp1a
OO 2160 bcs set1 ; yes, too big
AB 2170 inp1a stx len
AD 2180 jsr combyt ; get id
BG 2190 stx id
DG 2200 txa ; set status registers
CG 2210 bpl inp1c ; no optional string
KC 2220 jsr chkcom
NI 2230 jsr findvar ; find optional string
LE 2240 ldy #$02
AE 2250 inp1b lda ($47),y ; get descriptor for string
JK 2260 sta lenb,y
JJ 2270 dey
HG 2280 bpl inp1b
JM 2290 inp1c jsr priast ; print default input
IE 2300 inp1d lda #$00
ND 2310 sta $c6 ; clear keyboard buffer
LF 2320 sta curpos ; initial position of cursor
LP 2330 sta escflg ; escape flag = 0
BK 2340 inp2 ldy curpos
AM 2350 lda (beg),y ; get character under the cursor
LL 2360 sta iq ; save it
OK 2370 sta ir ; temporarily
JP 2380 inp3 jsr getkey ; get a keypress
DI 2390 sta $d7 ; save it temporarily
LN 2400 cmp #133 ; [f1]
BJ 2410 bne inp4
NK 2420 lda id
CL 2430 and #16 ; check id
FI 2440 beq inp3 ; not allowed
FO 2450 lda iq
PA 2460 ldy curpos ; restore character under cursor
PG 2470 sta (beg),y
IP 2480 ldx #$1 ; set escape flg
CG 2490 stx escflg
NA 2500 jmp return
PE 2510 inp4 cmp #32 ; [space]
IB 2520 beq inp5
LL 2530 cmp #160 ; [shifted-space]
LB 2540 bne inp6
LA 2550 inp5 lda #32 ; convert to a normal space
HP 2560 sta $d7
DG 2570 jmp gotit
EK 2580 inp6 cmp #48 ; [0]
IC 2590 bcc inp7
DL 2600 cmp #58 ; [9]+1
MH 2610 bcs inp7
FH 2620 lda id
GE 2630 and #2 ; check id
OB 2640 beq inp12 ; not allowed
DN 2650 jmp gotit ; [0-9]
CB 2660 inp7 cmp #65 ; [a]
NL 2670 bcc inp8a
ID 2680 cmp #91 ; [z]+1
BB 2690 bcs inp8a
JL 2700 inp8 lda id
CJ 2710 and #1 ; check id
OG 2720 beq inp12 ; not allowed
MC 2730 jmp gotit ; [a-z] or [shift a-shift z]
FP 2740 inp8a cmp #193 ; [shift a]
AN 2750 bcc inp9
OD 2760 cmp #219 ; [shift z]+1
EC 2770 bcs inp9
KO 2780 bcc inp8
PA 2790 inp9 cmp #157 ; [cursor left]
LD 2800 bne inp10
JI 2810 ldy curpos
JO 2820 beq inp3 ; can't cursor left
BG 2830 lda iq
BO 2840 sta (beg),y
ME 2850 dec curpos
EI 2860 jmp inp2
KE 2870 inp10 cmp #29 ; [cursor right]
MI 2880 bne inp11
JN 2890 ldy curpos
ED 2900 iny
LP 2910 cpy len
HJ 2920 beq inp3 ; can't cursor right
NC 2930 dey
PM 2940 lda iq
PE 2950 sta (beg),y
OF 2960 jsr check
JO 2970 inc curpos
MP 2980 jmp inp2
OP 2990 inp11 cmp #13 ; [return]
NN 3000 beq return
CH 3010 cmp #17 ; [cursor down]
```

```
CI  3020 beq down
MO  3030 cmp #145 ; [cursor up]
MG  3040 beq up
MB  3050 cmp #148 ; [insert]
HH  3060 beq insert
CJ  3070 cmp #46 ; [.]
NA  3080 beq decimal
DJ  3090 cmp #20 ; [delete]
JG  3100 bne inp12
LF  3110 jmp delete
NB  3120 inp12 bit id ; special characters allowed
LO  3130 bpl done ; no
HM  3140 ldy #$00
HA  3150 lda $d7
AP  3160 inp13 cmp (bst),y ; yes
BL  3170 bne inp14
FM  3180 jmp gotit
HD  3190 inp14 iny
BK  3200 cpy lenb
IN  3210 bne inp13
JF  3220 done jmp inp3 ; no other keys allowed
LO  3230 up ldx #$03
DI  3240 .byte $2c
IF  3250 down ldx #$02
FP  3260 lda id
NG  3270 and #8
MJ  3280 beq done
FL  3290 .byte $2c
PO  3300 return ldx #$01
HC  3310 lda id
LM  3320 and #64
FD  3330 beq ret1
AH  3340 jsr justr
JJ  3350 ret1 ldy curpos
DH  3360 lda iq
DP  3370 sta (beg),y
NG  3380 lda id
HF  3390 and #32 ; check for removing trailing spaces
LE  3400 beq ret4 ; no
NN  3410 ldy len
HB  3420 dey
NF  3430 ret2 lda (ast),y ; get character from a$
KO  3440 cmp #32 ; is it a space
AK  3450 bne ret3
PD  3460 dey
CN  3470 bpl ret2
OG  3480 ret3 iny
JF  3490 tya
PC  3500 ldy #$00
MH  3510 sta (varadr),y
HN  3520 ret4 txa ; type of return in location 780
GE  3530 pha
HH  3540 jsr priast
GG  3550 pla
KO  3560 ldx escflg ; get escape flag
ON  3570 rts
HK  3580 decimal lda id ; check id
NJ  3590 and #4
ON  3600 beq inp12 ; not allowed
LL  3610 jsr checkd ; check for decimal point
CJ  3620 beq cant ; decimal point already entered
HI  3630 jmp gotit
JG  3640 cant jmp inp3
GP  3650 insert ldy curpos
PJ  3660 lda iq
PB  3670 sta (beg),y
LO  3680 ldy len
FC  3690 dey
FB  3700 cpy curpos
EB  3710 beq cant
PG  3720 lda (ast),y
IJ  3730 cmp #32 ; is last character a space
ON  3740 bne cant ; can't insert
KF  3750 ins1 dey
KO  3760 lda (beg),y ; get screen code
EM  3770 pha ; save it
LK  3780 lda (ast),y
OK  3790 iny
LA  3800 sta (ast),y ; move character in string
KG  3810 pla
DL  3820 sta (beg),y ; move character on screen
BL  3830 dey
BK  3840 cpy curpos
OC  3850 bne ins1
LJ  3860 lda #32
ID  3870 sta (ast),y ; put space in string
LD  3880 ldx $c7
PG  3890 beq ins2
PA  3900 ora #$80
HO  3910 ins2 sta (beg),y ; put space on screen
IK  3920 jmp inp2
FK  3930 delete ldy curpos
MF  3940 bne del1
CJ  3950 iny ; cursor in first position
KI  3960 cpy len ; only one character
BA  3970 bne cant ; no, so can't delete
JA  3980 dey ; yes, so put a space
BO  3990 lda #32 ; in the first position
JG  4000 sta (beg),y
PM  4010 sta (ast),y
MA  4020 jmp inp2
JM  4030 del1 lda iq
BJ  4040 sta (beg),y
KP  4050 iny ; is cursor on last character
JH  4060 cpy len
GK  4070 bne del2 ; no
DL  4080 dey ; yes
BK  4090 lda (ast),y ; get last character
OH  4100 cmp #32 ; is it a space
HC  4110 beq del2 ; yes
LC  4120 inc curpos ; no
KG  4130 del2 ldy curpos
HO  4140 dey
CA  4150 lda (ast),y ; get character to delete
KN  4160 del3 iny
HO  4170 cpy len
BH  4180 beq del5
GI  4190 lda (ast),y ; character to replace
EO  4200 pha
NP  4210 lda (beg),y
HD  4220 dey
JJ  4230 ldx $c7
JK  4240 beq del4
NG  4250 ora #$80
GJ  4260 del4 sta (beg),y ; delete it on screen
GD  4270 pla
ID  4280 sta (ast),y ; delete it in string
CK  4290 iny
MM  4300 bne del3
NE  4310 del5 dey
HG  4320 lda #32
PA  4330 sta (ast),y
HA  4340 ldx $c7
PB  4350 beq del6
LN  4360 ora #$80
IJ  4370 del6 sta (beg),y
```

```
GE  4380 dec curpos
OH  4390 jmp inp2
IA  4400 gotit jsr check
JM  4410 ldy curpos
NP  4420 lda $d7
HO  4430 sta (ast),y ; put it in string
DP  4440 bmi got3
MN  4450 cmp #$60
LM  4460 bcc got1
NJ  4470 and #$df
LA  4480 bne got2
DF  4490 got1 and #$3f
MK  4500 got2 jmp got5
PH  4510 got3 and #$7f
NH  4520 cmp #$7f
DE  4530 bne got4
CE  4540 lda #$5e
HO  4550 got4 ora #$40
DF  4560 got5 ldx $c7
GI  4570 beq got6
HL  4580 ora #$80
JN  4590 got6 sta (beg),y
IN  4600 iny
PJ  4610 cpy len
GK  4620 bne got7
BN  4630 dey
EM  4640 got7 sty curpos
CI  4650 jmp inp2
KK  4660 ;
ON  4670 ; justify left
OL  4680 ;
JA  4690 tempm .byte 0
FB  4700 tempn .byte 0
MN  4710 ;
KC  4720 just1 ldy #$00
CN  4730 sty tempm
LG  4740 lda (ast),y
KF  4750 cmp #32
NG  4760 bne jus5 ; already justified
CJ  4770 jus1 iny
JE  4780 cpy len
MI  4790 beq jus5 ; all spaces
HK  4800 lda (ast),y
GJ  4810 cmp #32
MB  4820 beq jus1
HF  4830 sty tempn ; first non-space character
CH  4840 jus2 ldy tempm ; move left
HB  4850 sta (ast),y
HN  4860 inc tempn
AO  4870 inc tempm
LC  4880 ldy tempn
HL  4890 cpy len
EH  4900 beq jus3
FB  4910 lda (ast),y
PG  4920 bne jus2
OI  4930 beq jus2
DK  4940 jus3 ldy tempm ; rest are spaces
NN  4950 lda #32
IJ  4960 jus4 sta (ast),y
KE  4970 iny
BB  4980 cpy len
EJ  4990 bcc jus4
II  5000 jus5 rts
IA  5010 ;
BH  5020 ; justify right
MB  5030 ;
KF  5040 justr ldy len
FH  5050 dey
MB  5060 sty tempm
FL  5070 lda (ast),y
EK  5080 cmp #32
HD  5090 bne just5 ; already justified
KL  5100 just1 dey
MA  5110 bmi just5 ; all spaces
HO  5120 lda (ast),y
GN  5130 cmp #32
JB  5140 beq just1
HJ  5150 sty tempn ; first non-space character
BK  5160 just2 ldy tempm
HF  5170 sta (ast),y
BP  5180 dec tempm
MP  5190 dec tempn
LG  5200 ldy tempn
JF  5210 bmi just3
LE  5220 lda (ast),y
PF  5230 bne just2
OH  5240 beq just2
IN  5250 just3 ldy tempm ; rest are spaces
DB  5260 lda #32
IK  5270 just4 sta (ast),y
LF  5280 dey
PL  5290 bpl just4
KO  5300 just5 rts
ED  5310 ;
HK  5320 ; print string
IE  5330 ;
CC  5340 priast lda $d7
CG  5350 pha
LJ  5360 ldy yval
AK  5370 ldx xval
KH  5380 clc
IA  5390 jsr plot
LJ  5400 ldy #$00
EP  5410 pri1 lda (ast),y
FM  5420 jsr $ffd2
GB  5430 iny
NN  5440 cpy len
PF  5450 bne pri1
MN  5460 pla
FF  5470 sta $d7
EF  5480 rts
IO  5490 ;
GO  5500 ; check justify flag
MP  5510 ;
IN  5520 check bit justf
ON  5530 bmi ch1 ; already on
NN  5540 lda id
BI  5550 and #64
LL  5560 beq ch1 ; not allowed
GM  5570 jsr just1 ; justify string and
FH  5580 jsr priast ; print it
OM  5590 lda #$80 ; set flag
JA  5600 sta justf
PP  5610 ch1 rts
KG  5620 ;
LH  5630 ; check for decimal
OH  5640 ;
IJ  5650 checkd ldy len
HN  5660 dey
FP  5670 check1 lda (ast),y
KA  5680 cmp #46
FA  5690 beq check2 ; found one
PP  5700 dey
NE  5710 bpl check1
HO  5720 lda #$01 ; no decimal point
FI  5730 check2 rts
```

Listing 2:"input demo"

```
CE  100 if a=0 then a=1: load "input.obj",8,1
LN  110 print "{clr}{green}{14 right}INPUT DEMO"
BM  120 poke 53281,0: poke 53280,0: poke 53272,23
CG  130 pr=49152: in=49155: ret=780: esc=781
IA  140 b$=".": c$="-": d$="$"
HN  150 sys pr,1,4,"Name:"
BP  160 sys pr,1,6,"Address:"
JC  170 sys pr,1,8,"City:"
JJ  180 sys pr,23,8,"Phone:"
OG  190 sys pr,1,10,"Amount Owed: {rvs}{cyan}{10 spaces}{rvs off} - Dues"
OJ  200 sys pr,14,11,"{rvs}{10 spaces}{rvs off} - Disks
MK  210 sys pr,14,12,"{rvs}{10 spaces}{rvs off} - Magazines
FD  220 sys pr,14,13,"{red}{rvs}{10 spaces}{rvs off} - Total
BG  230 read na$,ad$,ci$,ph$,du$,di$,ma$,tl$
JP  240 data "Garry Kiziak{8 spaces}","2381 Duncaster Drive{5 spaces}"
IE  250 data "Burlington{5 spaces}","335-4837"
BO  260 data "{5 spaces}$0.00","{5 spaces}$3.50",
          "{4 spaces}$12.00","{4 spaces}$15.50"
AF 1000 print "{yellow}":sys in,7,4,na$,20,153,b$:if peek(esc) then 1160
HJ 1010 on peek(ret) goto 1020,1020,1140
KJ 1020 print "{yellow}":sys in,10,6,ad$,25,155,b$:if peek(esc) then 1160
MK 1030 on peek(ret) goto 1040,1040,1000
MI 1040 print "{yellow}":sys in,7,8,ci$,15,153,b$:if peek(esc) then 1160
OM 1050 on peek(ret) goto 1060,1060,1020
OJ 1060 print "{yellow}":sys in,30,8,ph$,8,154,c$:if peek(esc) then 1160
AP 1070 on peek(ret) goto 1080,1080,1040
DN 1080 print "{cyan}{rvs}";:sys in,14,10,du$,10,222,d$:if peek(esc) then 1160
HO 1090 on peek(ret) goto 1100,1100,1060
IL 1100 print "{cyan}{rvs}";:sys in,14,11,di$,10,222,d$:if peek(esc) then 1160
JA 1110 on peek(ret) goto 1120,1120,1080
IM 1120 print "{cyan}{rvs}";:sys in,14,12,ma$,10,222,d$:if peek(esc) then 1160
IB 1130 on peek(ret) goto 1140,1140,1100
CB 1140 print "{red}{rvs}";:sys in,14,13,tl$,10,222,d$:if peek(esc) then 1160
FB 1150 on peek(ret) goto 1000,1000,1120
BM 1160 sys pr,4,20,"{rvs off}{white}That's all there is to it!!!"
```

Listing 3: "input.obj maker"

```
AH  10 open 15,8,15,"s0:input.obj"
FN  20 open 1,8,1,"0:input.obj"
FO  30 print#1,chr$(0);chr$(192);
EP  40 for i=0 to 853
IE  50 read x
NK  60 print#1,chr$(x);
FC  70 next i
DC  80 close 1
BA  90 close 15
EG  100 end
DP 1000 data  76,  39, 192,  76,  95, 192,   0,   0,  32, 241
FO 1010 data 183, 224,  40, 176,  21, 142,   7, 192, 138,  72
EC 1020 data  32, 241, 183, 224,  25, 176,   9, 142,   6, 192
KO 1030 data 104, 168,  24,  76, 240, 255,  76,  72, 178,  32
JA 1040 data   8, 192,  32, 253, 174,  76, 164, 170, 173,  93
JD 1050 data 192,  73, 128, 141,  93, 192, 164, 253, 145, 251
LK 1060 data 169,  16, 141,  89, 192, 169, 255, 141,  88, 192
LL 1070 data  32, 228, 255, 208,  12, 206,  88, 192, 208, 246
CJ 1080 data 206,  89, 192, 208, 241, 240, 217,  96,   0,   0
AP 1090 data   0,   0,   0,   0,   0, 169,   0, 141,  90, 192
PC 1100 data  32,   8, 192,  24, 165, 209, 101, 211, 133, 251
PP 1110 data 165, 210, 105,   0, 133, 252,  32, 253, 174,  32
JF 1120 data 139, 176, 133,   5, 132,   6, 160,   2, 177,   5
JO 1130 data 153,   2,   0, 136,  16, 248, 165,   2, 240, 152
MI 1140 data  32, 241, 183, 138, 240, 146, 228,   2, 240,   4
LE 1150 data 144,   2, 176, 138, 134,   2,  32, 241, 183, 142
EM 1160 data  94, 192, 138,  16,  16,  32, 253, 174,  32, 139
AK 1170 data 176, 160,   2, 177,  71, 153, 178,   0, 136,  16
JM 1180 data 248,  32,  18, 195, 169,   0, 133, 198, 133, 253
AB 1190 data 141,  91, 192, 164, 253, 177, 251, 141,  92, 192
CA 1200 data 141,  93, 192,  32,  48, 192, 133, 215, 201, 133
AM 1210 data 208,  22, 173,  94, 192,  41,  16, 240, 240, 173
JC 1220 data  92, 192, 164, 253, 145, 251, 162,   1, 142,  91
FP 1230 data 192,  76, 146, 193, 201,  32, 240,   4, 201, 160
DM 1240 data 208,   7, 169,  32, 133, 215,  76, 100, 194, 201
DO 1250 data  48, 144,  14, 201,  58, 176,  10, 173,  94, 192
HL 1260 data  41,   2, 240, 101,  76, 100, 194, 201,  65, 144
OG 1270 data  14, 201,  91, 176,  10, 173,  94, 192,  41,   1
PM 1280 data 240,  83,  76, 100, 194, 201, 193, 144,   6, 201
LD 1290 data 219, 176,   2, 144, 236, 201, 157, 208,  14, 164
FG 1300 data 253, 240, 156, 173,  92, 192, 145, 251, 198, 253
NJ 1310 data  76, 193, 192, 201,  29, 208,  21, 164, 253, 200
FF 1320 data 196,   2, 240, 135, 136, 173,  92, 192, 145, 251
JP 1330 data  32,  47, 195, 230, 253,  76, 193, 192, 201,  13
AA 1340 data 240,  60, 201,  17, 240,  46, 201, 145, 240,  39
LI 1350 data 201, 148, 240, 120, 201,  46, 240,  98, 201,  20
JF 1360 data 208,   3,  76,  14, 194,  44,  94, 192,  16,  16
JB 1370 data 160,   0, 165, 215, 209, 179, 208,   3,  76, 100
JI 1380 data 194, 200, 196, 178, 208, 244,  76, 203, 192, 162
PD 1390 data   3,  44, 162,   2, 173,  94, 192,  41,   8, 240
HK 1400 data 241,  44, 162,   1, 173,  94, 192,  41,  64, 240
JH 1410 data   3,  32, 217, 194, 164, 253, 173,  92, 192, 145
OK 1420 data 251, 173,  94, 192,  41,  32, 240,  18, 164,   2
GJ 1430 data 136, 177,   3, 201,  32, 208,   3, 136,  16, 247
CP 1440 data 200, 152, 160,   0, 145,   5, 138,  72,  32,  18
BO 1450 data 195, 104, 174,  91, 192,  96, 173,  94, 192,  41
JA 1460 data   4, 240, 158,  32,  71, 195, 240,   3,  76, 100
AD 1470 data 194,  76, 203, 192, 164, 253, 173,  92, 192, 145
FL 1480 data 251, 164,   2, 136, 196, 253, 240, 239, 177,   3
FB 1490 data 201,  32, 208, 233, 136, 177, 251,  72, 177,   3
KJ 1500 data 200, 145,   3, 104, 145, 251, 136, 196, 253, 208
DH 1510 data 239, 169,  32, 145,   3, 166, 199, 240,   2,   9
PG 1520 data 128, 145, 251,  76, 193, 192, 164, 253, 208,  15
ND 1530 data 200, 196,   2, 208, 192, 136, 169,  32, 145, 251
FB 1540 data 145,   3,  76, 193, 192, 173,  92, 192, 145, 251
IF 1550 data 200, 196,   2, 208,   9, 136, 177,   3, 201,  32
LM 1560 data 240,   2, 230, 253, 164, 253, 136, 177,   3, 200
IA 1570 data 196,   2, 240,  20, 177,   3,  72, 177, 251, 136
ND 1580 data 166, 199, 240,   2,   9, 128, 145, 251, 104, 145
CN 1590 data   3, 200, 208, 231, 136, 169,  32, 145,   3, 166
KF 1600 data 199, 240,   2,   9, 128, 145, 251, 198, 253,  76
EG 1610 data 193, 192,  32,  47, 195, 164, 253, 165, 215, 145
OF 1620 data   3,  48,  13, 201,  96, 144,   4,  41, 223, 208
EH 1630 data   2,  41,  63,  76, 134, 194,  41, 127, 201, 127
IH 1640 data 208,   2, 169,  94,   9,  64, 166, 199, 240,   2
KD 1650 data   9, 128, 145, 251, 200, 196,   2, 208,   1, 136
LA 1660 data 132, 253,  76, 193, 192,   0,   0, 160,   0, 140
LI 1670 data 153, 194, 177,   3, 201,  32, 208,  50, 200, 196
JD 1680 data   2, 240,  45, 177,   3, 201,  32, 240, 245, 140
DN 1690 data 154, 194, 172, 153, 194, 145,   3, 238, 154, 194
LN 1700 data 238, 153, 194, 172, 154, 194, 196,   2, 240,   6
BI 1710 data 177,   3, 208, 234, 240, 232, 172, 153, 194, 169
HP 1720 data  32, 145,   3, 200, 196,   2, 144, 249,  96, 164
CG 1730 data   2, 136, 140, 153, 194, 177,   3, 201,  32, 208
BN 1740 data  44, 136,  48,  41, 177,   3, 201,  32, 240, 247
MP 1750 data 140, 154, 194, 172, 153, 194, 145,   3, 206, 153
DL 1760 data 194, 206, 154, 194, 172, 154, 194,  48,   6, 177
CB 1770 data   3, 208, 236, 240, 234, 172, 153, 194, 169,  32
JN 1780 data 145,   3, 136,  16, 251,  96, 165, 215,  72, 172
MO 1790 data   7, 192, 174,   6, 192,  24,  32, 240, 255, 160
BI 1800 data   0, 177,   3,  32, 210, 255, 200, 196,   2, 208
EP 1810 data 246, 104, 133, 215,  96,  44,  90, 192,  48,  18
JC 1820 data 173,  94, 192,  41,  64, 240,  11,  32, 155, 194
JD 1830 data  32,  18, 195, 169, 128, 141,  90, 192,  96, 164
OB 1840 data   2, 136, 177,   3, 201,  46, 240,   5, 136,  16
JB 1850 data 247, 169,   1,  96
```

Sprite Rotation

A New Twist

by Jim Frost

One *Transactor* every two months is not nearly enough for a confirmed ML addict like me, so I eventually bought a complete set of back issues. Between projects, if my wife isn't insisting I mow the lawn or fix the leaking faucets, the entire *Transactor* collection is reread for new programming ideas. A machine language version of Chris Zamara's Sprite Rotate (*Transactor*, Volume 5 Issue 1) seemed a suitable challenge so I decided to give it a try. The project took over a year of study, trial, and (mostly) error prior to successful completion. Along the way I learned to use ROM trig routines, unravelled the mysteries of floating point math and mastered some of the complexities of graphic rotation.

Using the Rotate Routine

The Rotate program included with this article will spin a complete sprite in under a second, fast enough to allow use from BASIC. Use the syntax SYS 49152, SA,DA,CX,CY,RA. SA and RA are the source and destination addresses of the target sprite. CX and CY are the vertical and horizontal axes of rotation, respectively, with rows and columns numbered from zero in the upper left corner. RA is the radian angle of rotation. The rotate routine requires that the source sprite be memory resident and will create one rotated copy per call. To reduce program length, variable limits are not tested.

Rotation calculations are performed on set pixels only, allowing small sprites to be rotated very quickly. To prevent annoying flicker when rotating large sprites, change sprite pointers only after the rotation is complete. Because the rotated sprite is rounded to pixel boundaries, an exact representation is rarely possible. Depending on the shape and detail of your sprite, some rotation angles provide better results than others. Experiment and use the angles that work best.

If you want to use Rotate in your ML programs, load $FB and $FC with the source sprite address, load $FD and FE with the destination sprite address, and load variables CX, CY, SINM, COSM, SGNSIN and SGNCOS with the desired values prior to calling. SGNSIN and SGNCOS are trig function signs. These should be set to zero for positive functions or one for negative functions. SGNM and COSM must be 256 times the actual SGN or COS values (use $FF for 1). With variables set, enter Rotate at the label MLENT.

Quantization

As I developed the sprite rotate program I encountered several unplanned difficulties, primarily due to rounding inaccuracies and quantization limits. Quantization means that a quantity exists in integer steps only, with no possible in-between values. Discounting the possibility of a sharp knife, seeds in an orange are quantized. Your orange might have one, two or five seeds, mine probably 20 or more but no possibility of 13.75. Pixel positions on a sprite or bit map screen are also quantized. We can draw a spot at the X,Y position 12,7 but not at pixel position 12.73,7.42. The rotation equations (see assembly listing) allow a precise calculation of exactly where a rotated pixel belongs. Quantization, however, prevents perfect pixel placement, leading to distortion of the rotated image and occasional holes. In my rotate routine, holes were minimized by detecting adjacent bits along the X axis and plotting the point midway between them. The current version of sprite rotate still shows a few holes when a solid (all bytes $FF) sprite is rotated to angles near 45 degrees. Without the extra plotting, the results resemble Swiss Cheese.

Understanding the Routine

With experience, expressing integers in ML is easy, but how can fractions be handled? In everyday math, the decimal point separates integer and fractional quantities, with numbers to the right of the decimal weighted by 10E-1, 10E-2 and so on. The same rules apply in binary. While bit zero is normally weighted by 2E0, this convention may be changed as desired, provided that values are correctly used throughout the program.

An alternate way of looking at binary fractions is to apply scaling. For example, rather than trying to express one half directly in binary, multiply .5 by 256 and use the resulting 128 ($80) in your program. Results are 256 times too large, but can be rescaled after all mathematics are completed. Scaling is not a second method; it's simply an alternate approach to understanding the technique.

After several months of experimenting with rotation, I suddenly realized that massive multiplication is not required. Since rotation equations are linear, the effects of X and Y changes are independent. This realization led to calculating a lookup

table by addition after multiplying to locate the first point. The current routine uses lookup tables for X only, as speed improvements in Y were not dramatic. The multiply routine is unusual in its handling of signed numbers. If you are planning a program where both positive and negative variables can occur, checking this portion of code may provide some new ideas.

Several approximations used in the sprite rotate routines are permitted by the small size (21 by 24 pixels) of sprites. Any sprite pixel position can be expressed in five bits, allowing a truncated multiplication. Sines and cosines can be approximated to an accuracy of one part in 256 in a single byte (MSB = 2E-1). For sprite-sized objects higher accuracy is unnecessary. These two simplifications reduce code requirements and speed calculations considerably.

The present routine accomplishes my original goals; however, I'm not completely satisfied. I'm still researching and analyzing to find the ultimate rotation algorithm. If you have questions on the current routine or suggestions on better methods, feel free to drop me a line. I have one idea I'd like to try right now, but first I'd better finish mowing the lawn.

Listing 1: BASIC demo program for the sprite rotate routine.

```
LL  10 rem revolving gun turret demonstrates
AA  20 rem sprite rotation and "holes"
NI  30 if m=0 then m=1: load "rotate.o",8,1
PC  40 poke 53280,0: poke 53281,0: print"[clr]"
LL  50 sp=130: poke 2040,sp
PA  60 x=55350: poke x,1: vic=53248
LB  70 poke vic,40: poke vic+1,200
DC  80 poke vic+21,1
MO  90 for i=0 to 62: read a
OB  100 poke 8320+i,a: next
OF  110 ss=8320: k=0: cx=12: cy=10
HP  120 for i=1 to 32
EE  130 ra=2*[pi]/32*i: ds=8384+64*k
LM  140 sys 49152,ss,ds,cx,cy,ra
FL  150 poke 2040,131+k
FF  160 k=k+1 and 1
OK  170 next
CB  180 data  0,   0,   0
MB  190 data  0,   0,   0
GC  200 data  0,   0,   0
LP  210 data  0,  36,   0
FA  220 data  0,  36,   0
PA  230 data  0,  36,   0
JB  240 data  0,  36,   0
DC  250 data  0,  36,   0
NC  260 data  0,  36,   0
HD  270 data  0,  36,   0
BE  280 data  0,  36,   0
EJ  290 data 31, 255, 248
OJ  300 data 31, 255, 248
IK  310 data 31, 255, 248
CL  320 data 31, 255, 248
ML  330 data 31, 255, 248
GM  340 data 31, 255, 248
AN  350 data 31, 255, 248
EM  360 data 15, 255, 240
KO  370 data  7, 255, 224
KN  380 data  0,   0,   0
```

Listing 2: Generator program to create "rotate.o" on disk.

```
CI  1000 rem  generator for "rotate.o"
BB  1010 nd$="rotate.o": rem  name of program
ML  1020 nd=824: sa=49152: ch=99925
OG  1030 for i=1 to nd: read x
IK  1040 ch=ch-x: next
ML  1050 if ch<>0 then print"data error": stop
JN  1060 print"data ok, now creating file": print
GE  1070 restore
LP  1080 open 8,8,1,"0:"+f$
GN  1090 print#8,chr$(sa/256)chr$(sa-int(sa/256));
EL  1100 for i=1 to nd: read x
GN  1110 print#8,chr$(x);: next
IE  1120 close 8
GG  1130 print"prg file '";f$;"' created..."
GP  1140 print"this generator no longer needed."
CP  1150 :
JN  1000 data  32,  40, 194,  32,   1, 184, 132, 251
PN  1010 data 133, 252,  32,  40, 194,  32,   1, 184
LH  1020 data 132, 253, 133, 254,  32,  40, 194,  32
LA  1030 data   1, 184, 140, 192, 194,  32,  40, 194
PM  1040 data  32,   1, 184, 140, 193, 194,  32,  40
IG  1050 data 194, 162, 181, 160, 194,  32, 212, 187
FF  1060 data  32, 107, 226,  32, 137, 194, 141, 186
AM  1070 data 194, 142, 194, 194, 169, 181, 160, 194
KL  1080 data  32, 162, 187,  32, 100, 226,  32, 137
LM  1090 data 194, 141, 187, 194, 142, 195, 194, 160
FI  1100 data  63, 169,   0, 145, 253, 136,  16, 251
AH  1110 data 162,   0, 142, 196, 194, 174, 195, 194
IM  1120 data 172, 187, 194, 169,   1,  32,  47, 194
PL  1130 data 141, 207, 194, 140, 208, 194, 174, 195
DK  1140 data 194, 173, 187, 194,  74, 168, 169,   1
CH  1150 data  32,  47, 194, 141, 209, 194, 140, 210
GL  1160 data 194, 162,   1, 142, 196, 194, 174, 194
JP  1170 data 194, 172, 186, 194, 169,   1,  32,  47
MO  1180 data 194, 141, 211, 194, 140, 212, 194, 174
JE  1190 data 194, 194, 173, 186, 194,  74, 168, 169
NN  1200 data   1,  32,  47, 194, 141, 213, 194, 140
PN  1210 data 214, 194, 162,   1, 142, 196, 194, 174
FH  1220 data 195, 194, 172, 187, 194, 173, 192, 194
JM  1230 data  32,  47, 194, 141,   8, 195, 140,  32
BH  1240 data 195, 206, 196, 194, 174, 194, 194, 173
CH  1250 data 192, 194, 172, 186, 194,  32,  47, 194
KC  1260 data 141, 216, 194, 140, 240, 194, 162,   0
LK  1270 data  24, 189,   8, 195, 109, 207, 194, 157
AJ  1280 data   9, 195, 189,  32, 195, 109, 208, 194
BG  1290 data 157,  33, 195,  24, 189, 216, 194, 109
PH  1300 data 211, 194, 157, 217, 194, 189, 240, 194
FF  1310 data 109, 212, 194, 157, 241, 194, 232, 224
FH  1320 data  23, 208, 213, 169, 255, 141, 202, 194
MI  1330 data 141, 197, 194, 238, 197, 194,  32, 218
ND  1340 data 193, 162,   0, 238, 202, 194, 172, 202
EP  1350 data 194, 177, 251, 160,   8,  14, 215, 194
NH  1360 data  42, 176,  16, 232, 136, 208, 249, 224
JL  1370 data  24, 208, 232, 173, 202, 194, 201,  62
CB  1380 data 208, 217,  96, 133,  98, 134,  99, 132
JB  1390 data 100,  24, 189,   8, 195, 109, 188, 194
ON  1400 data 141, 205, 194, 189,  32, 195, 109, 189
GA  1410 data 194, 141, 199, 194,  24, 189, 216, 194
```

```
FO  1420 data 109, 190, 194, 141, 206, 194, 189, 240
HO  1430 data 194, 109, 191, 194, 141, 200, 194, 141
OL  1440 data 201, 194,  48,  19, 201,  21, 176,  15
JF  1450 data 173, 199, 194,  48,  10, 201,  24, 176
KA  1460 data   6, 173, 200, 194,  32, 178, 193, 165
DB  1470 data  98,  16,  46,  24, 173, 205, 194, 109
OC  1480 data 209, 194, 173, 199, 194, 109, 210, 194
NI  1490 data 141, 199, 194,  48,  28, 201,  24, 176
BE  1500 data  24, 173, 206, 194, 109, 213, 194, 173
AB  1510 data 201, 194, 109, 214, 194, 141, 200, 194
LI  1520 data  48,   7, 201,  21, 176,   3,  32, 178
DO  1530 data 193, 165,  98, 166,  99, 164, 100,  76
FP  1540 data  35, 193,  10, 109, 200, 194, 141, 200
KN  1550 data 194, 173, 199, 194,  41,   7, 170, 173
FG  1560 data 199, 194,  74,  74,  74,  24, 109, 200
GH  1570 data 194, 168, 189, 210, 193,  17, 253, 145
LI  1580 data 253,  96, 128,  64,  32,  16,   8,   4
BE  1590 data   2,   1, 162,   1, 142, 196, 194, 174
EM  1600 data 194, 194, 172, 186, 194,  56, 173, 193
EK  1610 data 194, 237, 197, 194, 141, 198, 194,  32
DN  1620 data  47, 194, 105, 128, 141, 188, 194, 152
GO  1630 data 109, 192, 194, 141, 189, 194, 174, 195
EN  1640 data 194, 172, 187, 194, 173, 198, 194,  32
EN  1650 data  47, 194, 105, 128, 141, 190, 194, 152
LN  1660 data 109, 193, 194, 141, 191, 194, 172, 202
PI  1670 data 194, 200, 200, 177, 251,  42, 200, 177
JP  1680 data 251, 106,  41, 192, 141, 215, 194,  96
DD  1690 data  32, 253, 174,  32, 158, 173,  96, 140
GN  1700 data 204, 194, 142, 180, 194, 160,   0, 140
OE  1710 data 178, 194, 162,   1, 201,   0,  16,   6
ML  1720 data  24,  73, 255, 105,   1, 232, 202, 142
GC  1730 data 179, 194,  10,  10,  10, 141, 203, 194
NO  1740 data 169,   0, 162,   5,  10,  46, 178, 194
CB  1750 data  14, 203, 194, 144,   9,  24, 109, 204
JA  1760 data 194, 144,   3, 238, 178, 194, 202, 208
KG  1770 data 235, 170, 173, 178, 194, 168, 173, 180
LJ  1780 data 194,  77, 179, 194,  77, 196, 194, 240
PO  1790 data  13, 138,  73, 255,  24, 105,   1, 170
KE  1800 data 152,  73, 255, 105,   0, 168, 138,  24
AL  1810 data  96, 165,  97, 201, 129, 240,  18, 201
JB  1820 data 128, 240,  20, 201, 120, 144,  13, 170
IF  1830 data 165,  98,  74, 232, 224, 128, 208, 250
JI  1840 data  44, 169, 255,  44, 169,   0,  44, 165
NP  1850 data  98, 162,   0,   6, 102, 144,   2, 162
CK  1860 data   1,  96,   0,   0,   0,   0,   0,   0
MB  1870 data   0,   0,   0,   0,   0,   0,   0,   0
GC  1880 data   0,   0,   0,   0,   0,   0,   0,   0
AD  1890 data   0,   0,   0,   0,   0,   0,   0,   0
KD  1900 data   0,   0,   0,   0,   0,   0,   0,   0
EE  1910 data   0,   0,   0,   0,   0,   0,   0,   0
OE  1920 data   0,   0,   0,   0,   0,   0,   0,   0
IF  1930 data   0,   0,   0,   0,   0,   0,   0,   0
CG  1940 data   0,   0,   0,   0,   0,   0,   0,   0
MG  1950 data   0,   0,   0,   0,   0,   0,   0,   0
GH  1960 data   0,   0,   0,   0,   0,   0,   0,   0
AI  1970 data   0,   0,   0,   0,   0,   0,   0,   0
KI  1980 data   0,   0,   0,   0,   0,   0,   0,   0
EJ  1990 data   0,   0,   0,   0,   0,   0,   0,   0
OJ  2000 data   0,   0,   0,   0,   0,   0,   0,   0
IK  2010 data   0,   0,   0,   0,   0,   0,   0,   0
CL  2020 data   0,   0,   0,   0,   0,   0,   0,   0
```

Listing 3: Merlin-format assembler source code.

```
* sprite rotate  rev 6 jan 88
* from basic program by chris zamara
* transactor vol5 #1
* to use sys 49152,ss,ds,cx,cy,ra
*
* jim frost
* 4740 harbinson ave
* la mesa ca 92041
**********************************

* equates

facsgn   = $66
facexp   = $61
facm0    = $62
stfalxy  = $bbd4
ldfalay  = $bba2
sine     = $e26b
cosine   = $e264
chkcom   = $aefd
evalexp  = $ad9e
fltfix   = $b801

         org $c000

* basic entry point

         jsr eval        ;fetch source sprite address
         jsr fltfix      ;convert to integer
         sty $fb         ;save for drawing
         sta $fc

         jsr eval        ;fetch dest sprite address
         jsr fltfix      ;convert to integer
         sty $fd         ;save for drawing
         sta $fe

         jsr eval        ;fetch cx
         jsr fltfix      ;convert to integer
         sty cx          ;and save

         jsr eval        ;fetch cy
         jsr fltfix      ;convert to integer
         sty cy          ;and save

         jsr eval        ;arg in fac1

         ldx #<arg
         ldy #>arg
         jsr stfalxy     ;store fp argument

         jsr sine        ;sina in fac1
         jsr normize     ;convert to one byte

         sta sinm        ;save sine
         stx sgnsin      ;and sign

         lda #<arg
         ldy #>arg
         jsr ldfalay     ;move argument to fac1

         jsr cosine      ;cosa in fac1
         jsr normize     ;convert to one byte
         sta cosm        ;save cos
         stx sgncos      ;and sign

* clear destination sprite memory from machine
* language enter here with variables set

mlent    ldy #$3f        ;64 bytes to clear
         lda #$00

cdest    sta ($fd),y     ;clear byte
         dey             ;decrement count
         bpl cdest       ;loop till 64 bytes cleared

* calculate table of portions of x2 and y2 due to
```

```
* x position across sprite

* convert one byte sine and cosine to two byte
* signed integer with sign adjusted for adding
* to current value x2(x),y2(x). as x increases
* note that the rotated x (x1) = x center (cx)
* - x so that as x increases x1 decreases
* resulting in signs the opposite of initial
* expectations

* calculate signed cos terms

        ldx #$00
        stx neg         ;don't flip sign

        ldx sgncos
        ldy cosm
        lda #$01        ;multiply by 1

        jsr mult        ;return two byte cos in ay
        sta cosl        ;save low byte
        sty cosh        ;and high

* calculate signed cos terms for half pixel step
* (used to minimize rounding and quantization errors)

        ldx sgncos
        lda cosm
        lsr             ;a has cos/2
        tay             ;y has cos/2
        lda #$01        ;multiply by 1

        jsr mult        ;return two byte cos/2 in ay
        sta hcosl
        sty hcosh

* calculate signed sin terms

        ldx #$01
        stx neg         ;this time flip sign

        ldx sgnsin
        ldy sinm
        lda #$01        ;multiply by 1

        jsr mult        ;return two byte -sin in ay
        sta sinl        ;save low byte
        sty sinh        ;and high

* calculate signed sin terms for half pixel step
* (used to minimize rounding and quantization errors)

        ldx sgnsin
        lda sinm
        lsr             ;a has sin/2
        tay             ;y has sin/2
        lda #$01        ;multiply by 1

        jsr mult        ;return two byte sin/2 in ay
        sta hsinl
        sty hsinh

* calculate first table entry

        ldx #$01        ;set sign flag
        stx neg         ;sign mult negative

        ldx sgncos      ;sign of cosine
        ldy cosm        ;cosine

        lda cx          ;x center of rotation

        jsr mult        ;return xlcosa in ay
        sta tbxl        ;and stash first table value
        sty tbxh        ;low and high bytes

        dec neg         ;clear neg flag

        ldx sgnsin      ;sign of sine
        lda cx          ;x0 minus center of rotation
        ldy sinm        ;one byte sine

        jsr mult        ;return xlsina in ay
        sta tbyl        ;and stash first table values
        sty tbyh        ;low and high bytes

* add terms to form remainder of table

        ldx #$00        ;table pointer

tbx2y2  clc
        lda tbxl,x
        adc cosl
        sta tbxl+1,x

        lda tbxh,x
        adc cosh
        sta tbxh+1,x

        clc
        lda tbyl,x
        adc sinl
        sta tbyl+1,x

        lda tbyh,x
        adc sinh
        sta tbyh+1,x

        inx
        cpx #23         ;finished 24th element?
        bne tbx2y2      ;no - loop til done

* rotate sprite after calculating new positions

        lda #$ff
        sta bcount
        sta y0

nxtrow  inc y0          ;on first pass y=0
        jsr newrow      ;calculate y based parameters

        ldx #$00        ;start each row at left

nxtbyte inc bcount      ;byte counter

        ldy bcount      ;index to byte

        lda ($fb),y     ;get byte

        ldy #$08        ;8 bits per byte!
        asl adjbyt      ;shift msb to carry

* at last pass through shift, bit 7 of adjbyt is in bit 7
* of byte being tested

shift   rol             ;shift next bit to carry
        bcs spinbit     ;if empty don't rotate

shift2  inx             ;next column
        dey             ;last bit?
        bne shift

        cpx #24         ;last x?
        bne nxtbyte     ;no -  try another byte

        lda bcount      ;byte count
        cmp #$3e        ;done all 63?
        bne nxtrow      ;no - do another row

        rts             ;back to basic

*=======================================================
* calculate new bit positions.  if values on sprite
* grid plot them

* calculate x2=int(-ylsina-xlcosa+cx)

spinbit sta $62         ;save current test byte
        stx $63         ;save sprite x
        sty $64         ;save bit count

        clc
        lda tbxl,x      ;get x cos low byte
```

```
        adc ysin1       ;add -ysina low byte for round
        sta x21         ;save for half pixel calc

        lda tbxh,x      ;now add x cos high byte
        adc ysinh       ;to -ysina high byte
        sta x2          ;save integer x2 for plotting

*integer x2 now in a - fractional x2 in x

*calculate y2=int(+x1sina-y1cosa+cy)

        clc
        lda tby1,x      ;get x sin low byte
        adc ycos1       ;add -ysina for round
        sta y21         ;save for half pixel calc

        lda tbyh,x      ;now add xsin high byte
        adc ycosh       ;to -ycosa high byte
        sta y2          ;save integer (lost in plot)
        sta y2h         ;and a second copy for later

*test out of range y and x

        bmi toobig      ;if negative
        cmp #21         ;or larger than 24
        bcs toobig      ;skip other calculations

        lda x2
        bmi toobig      ;if negative
        cmp #24         ;or larger than 24
        bcs toobig      ;skip other calculations

        lda y2
        jsr plot        ;plot on destination sprite

* with current bit in carry, bit 7 is next adjacent bit.
* if neg flag set, there are two adjacent bits, so plot
* half pixel between them

toobig  lda $62         ;retrieve test byte
        bpl noplot      ;unless bit 7 set no plot

        clc
        lda x21
        adc hcos1       ;add half cos low

        lda x2
        adc hcosh       ;add integer ycos
        sta x2          ;save integer x2 for plotting

        bmi noplot      ;if negative
        cmp #24         ;or larger than 24
        bcs noplot      ;skip other calculations

        lda y21         ;carry always clear
        adc hsin1       ;add half sin

        lda y2h         ;copy of original y2
        adc hsinh       ;and add -ysina high byte
        sta y2          ;save integer

        bmi noplot      ;if negative
        cmp #21         ;or larger than 24
        bcs noplot      ;skip other calculations

        jsr plot        ;plot on destination sprite

noplot  lda $62         ;retrieve test byte
        ldx $63         ;retrieve sprite x
        ldy $64         ;retrieve bit count
        jmp shift2      ;back to test loop

* x and y calculated - plot on destination sprite

plot    asl             ;a=2*y2 - no carry guaranteed
        adc y2          ;a=3*y2
        sta y2          ;save for next calculation

        lda x2          ;get new x value
        and #$07        ;trash high nibble
        tax             ;save pointer to bittab

        lda x2
        lsr
        lsr
        lsr             ;a=x2/8

        clc
        adc y2          ;add bytes from x to 3*y
        tay             ;pointer to sprite row

        lda bitmask,x   ;get value of bit to set
        ora ($fd),y     ;or new bit with current one
        sta ($fd),y     ;save with new bit set

        rts

bitmask dfb %10000000
        dfb %01000000
        dfb %00100000
        dfb %00010000
        dfb %00001000
        dfb %00000100
        dfb %00000010
        dfb %00000001

*================================================
* handle all y related calculations one time for each
* sprite row rotated

newrow  ldx #$01        ;set flag
        stx neg         ;flip sign of product

        ldx sgnsin      ;sign of sine
        ldy sinm        ;one byte sine

        sec
        lda cy          ;y center of rotation
        sbc y0          ;subtract current y
        sta y1          ;value y to rotate

        jsr mult        ;return -y1sina in ay

        adc #$80        ;half round
        sta ysin1       ;save fractional part

        tya             ;get high (integer) byte
        adc cx          ;and add offset
        sta ysinh       ;save integer part

        ldx sgncos      ;sign of cosine
        ldy cosm        ;one byte cosine
        lda y1
        jsr mult        ;return -y1cosa in ay

        adc #$80        ;half round
        sta ycos1

        tya             ;get high (integer) byte
        adc cy          ;add in offset
        sta ycosh

* arrange bits to flag adjacent bit pairs between
* bytes of each row

        ldy bcount
        iny
        iny

        lda ($fb),y     ;grab middle byte of row
        rol             ;shift bit 7 to carry

        iny
        lda ($fb),y     ;grab last byte of row
        ror             ;shift carry to bit 7
        and #%11000000  ;bits of 'a' (1 to r) msb mid,
        sta adjbyt      ;msb last, trash

        rts

*================================================

eval    jsr chkcom
```

```
        jsr evalexp
        rts

*=============================================================

* multiply one byte trig function in y by x1 or y1 value
* (0 to 23) in a. use neg and sign of trig function in x
* to determine sign of product

mult    sty temp1       ;save value trig function
        stx msign       ;and sign
        ldy #$00        ;clear work space
        sty reshi
        ldx #$01        ;guess sign neg

* convert acc value to absolute value in a. check
* msign, neg and sign of acc value and set flag
* showing sign of product for multiply

        cmp #$00        ;set flags on a
        bpl apos        ;branch if positive

        clc             ;convert negative
        eor #$ff        ;value to positive
        adc #$01

        inx             ;adjust for next instruction

apos    dex             ;set sign flag positive
        stx psign       ;save sign flag

        asl             ;since twice sprite width is
        asl             ;24 max bits, 6, 7 and 8 are
        asl             ;always zero - trash them

        sta temp2       ;and save result

        lda #$00        ;clear lsb of product
        ldx #$05        ;five bits to multiply

shiftm  asl             ;shift product low byte
        rol reshi       ;and high

        asl temp2       ;shift msb to carry
        bcc nobitm      ;if no carry don't add

        clc
        adc temp1       ;else add to accumulator low
        bcc nobitm      ;bit and high bit if required

        inc reshi

nobitm  dex             ;decrement counter
        bne shiftm      ;and loop till six bits

        tax             ;hold result low
        lda reshi

*determine sign of product

        tay             ;save reshi for following
        lda msign
        eor psign
        eor neg         ;acc zero if negs cancel
        beq mdone       ;if positive

        txa             ;else recover low byte
        eor #$ff        ;flip bits
        clc
        adc #$01        ;negate low byte
        tax             ;and save

        tya             ;recover high byte
        eor #$ff        ;flip bits
        adc #$00        ;complete negation

        tay             ;msb in y

mdone   txa             ;lsb in a

        clc             ;for next add
        rts             ;lsb in a msb in y

*=============================================================

* shift facl value into a single
* byte with bit 7 value 2e-1

normize lda facexp
        cmp #$81        ;if exp is 81 value is 1
        beq val1

        cmp #$80        ;if exp is 80 no shift needed
        beq valok

        cmp #$78        ;if exp is < 78 value is zero
        bcc val0

        tax             ;move exponent to x
        lda facm0       ;get fac msb

nor1    lsr             ;shift bits and increase exp
        inx             ;until bit 7 has value of
        cpx #$80        ;2e-1
        bne nor1

        hex 2c          ;skip next instruction
val1    lda #$ff        ;1/256 less than 1

        hex 2c          ;skip next instruction
val0    lda #$00

        hex 2c          ;skip next instruction
valok   lda facm0

        ldx #$00        ;set x for positive
        asl facsgn      ;shift neg bit to carry
        bcc zsgn        ;if no carry, sign is pos

        ldx #$01        ;set x for negative

zsgn    rts

* variables for rotate

reshi   ds 01           ;high byte multiply result
psign   ds 01           ;sign of multiply product
msign   ds 01           ;sign of trig funct this mult
arg     ds 05           ;floating point value argument
sinm    ds 01           ;sina in multiply form
cosm    ds 01           ;cosa in multiply form
ysinl   ds 01           ;fractional part y*sina
ysinh   ds 01           ;integer part
ycosl   ds 01           ;fractional part y*cosa
ycosh   ds 01           ;integer part
cx      ds 01           ;x center of rotation
cy      ds 01           ;y center of rotation
sgnsin  ds 01           ;sign of sine term
sgncos  ds 01           ;sign of cos term
neg     ds 01           ;flag - value is negative
y0      ds 01           ;column count
y1      ds 01           ;y offset from cy
x2      ds 01           ;rotated x position
y2      ds 01           ;rotated y position
y2h     ds 01           ;copy of y2
bcount  ds 01           ;bit count
temp2   ds 01
temp1   ds 01
x21     ds 01
y21     ds 01
cosl    ds 01
cosh    ds 01
hcosl   ds 01
hcosh   ds 01
sinl    ds 01
sinh    ds 01
hsinl   ds 01
hsinh   ds 01
adjbyt  ds 01
tbyl    ds 24
tbyh    ds 24
tbxl    ds 24
tbxh    ds 24
```

Structured DATA and Seeding RND

Consult the oracle inside your computer

by Audrys Vilkas

The program that accompanies this article was written in Commodore BASIC and is an exercise in using structured data statements and seeding the random function RND provided by the BASIC interpreter. I will call it the Hexagram Program for reasons which will soon be clear. This program may be embellished with many "not too difficult to implement" subroutines, providing the reader with his or her own version.

Motivation

In the September 1986 issue of *Transactor* (Volume 7, Issue 2), there is an interesting little tidbit called "Animals: An Exercise in Artificial Intelligence", by Chris Zamara. In it he constructs a data base which "increases its knowledge as it is used..." by user interaction with the program. Questions are asked by the machine and the user's answers are stored in a record to be referenced later as the program matures.

On the other hand, in the May 10th, 1986 issue of *Science News* there is an article "Inside Averages" by Ivars Peterson, in which he discusses Diaconis' analysis of syllable patterns in Plato's books. Using techniques that depend on certain averages being known though original data are missing, and using certain statistical techniques applied to these so-called hidden averages, Diaconis is able to conclude that Plato wrote his books "top to bottom". These techniques are applied in such technology as X-ray tomography and side-view radar. I will not go into any technical detail on these subjects, but I will instead provide the reader with the gist of the Hexagram Program, which is a little more 'light-weight'.

This program is the problem of "Animals" somewhat in reverse. That is, the user begins asking the questions and the machine responds with a pseudo-random answer! Whether the answer is applicable to the question will be left for the user to decide. At first glance this may seem a bit outrageous but for now please bear with me.

The material for the Hexagram routine is rooted deeply in history and comes from what is known today as the *I Ching*. The *I Ching* (or *Chou I*) is a collection of symbols and writings of very great antiquity, at least 3000 years old; its origins may go back further still. Confucius referred to the document as "very old" 2500 years ago.

Some historical background will be presented below, but we will first explain what a hexagram is.

Reading the Hexagrams

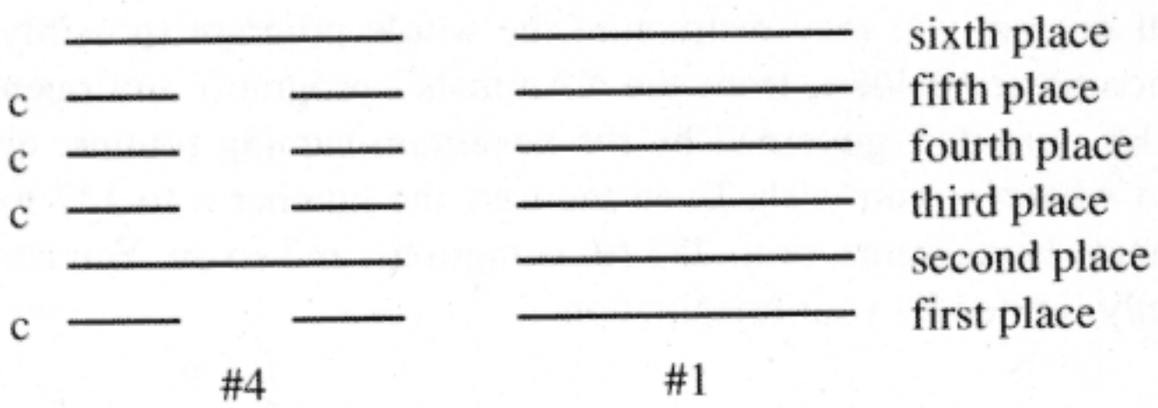
All hexagrams are composed of two trigrams (an upper and a lower) chosen from the following eight basic trigrams:

———————	Ch'ien
———————	Heaven, sky, cold, creative,
———————	father, active, strong, firm
——— ———	Tui
———————	Lake, marsh, rain, autumn,
———————	joyful, youngest daughter
———————	Li
——— ———	Fire, lightning, sun, summer,
———————	beautiful, middle daughter
——— ———	Che^n
——— ———	Thunder, spring, arousing,
———————	moving, active, eldest son
——— ———	K'an
———————	Water, cloud, moon, winter,
——— ———	dangerous, middle son
——— ———	K'un
——— ———	Earth, heat, receptive,
——— ———	yielding, dark, mother
———————	Sun
———————	Wind, wood, gentle,
——— ———	penetrating, eldest daughter
———————	Ke^n
——— ———	Mountain, thunder, stubborn,
——— ———	perverse, youngest son

Juxtaposing any two of the above trigrams produces a hexagram. There are additional sets of attributes and structure imposed on the hexagrams from which much meaning is derived, and over the centuries these have evolved into the associated

texts. These structures are complicated and we will not go into them here. There are four basic principles worth noting though: they are "The Great Yin", "The Lesser Yang", "The Lesser Yin" and "The Great Yang". Their mysterious polarities determine whether the lines in a hexagram are changing or not. Thus, the concept of a distinct hexagram pair is arrived at when there is a changing line.

If one receives a changing line, a Hexagram Generating Program could maybe highlight the line and mark it with a 'c' to indicate the change. This is the line to note when reading the hexagram's associated text, which could be titled "lines". The hexagrams are numbered from the bottom up, starting at line one (at the bottom) and going up to line six (at the top). For example, suppose we ask the question "How many times may I ask the same question?" and we get the following two hexagrams, #4 ("Youthful Folly") and #1 ("The Creative"):



The first hexagram (#4) is composed of "yin" lines except for a "yang" line in the second and sixth places. There are four dynamic yin (in the first, third, fourth and fifth places). These are the lines, in this case, which yield a distinct second hexagram. When one reads the hexagram, in addition to reading The Image and The Judgement, in this case, one also reads the changing lines in hexagram #4. (Note that hexagram #4 concerns the repeated asking of the same question - a "logical glitch".) The second hexagram - #1, The Creative - is also read but no text associated with the lines needs to be read. Of course, one may not receive any changing lines, so only The Image and The Judgement are to be read and the hexagram pair is nondistinct. Traditionally, if only one change occurs in a hexagram you then don't read the second hexagram. I will not follow that convention here.

Some Historical Background

Bernhard Kalgren, in his *Sound And Symbol*, writes of the legend:

"Long, long ago, in the golden age, there was a dragon horse which came out of the Yellow River with curious symbols traced upon its back, and revealed them to Fu-hsi (the first of China's legendary primeval emperors). This potentate copied them and thus acquired the mystical characters which later became the skeleton of the I King (now I Ching), the Canon of Changes, one of the Five Canons."

The *Book of Changes* consists of 64 hexagrams, and has a historiographical nature. According to Iulian K. Shchutskii, a Russian sinologist, the *I Ching* was basically a divinatory text that began taking on a philosophical countenance after many centuries of being appended by the commentary schools (in which, by the way, Confucius played no direct part). The *Book* was then employed by politicians in China and Japan. Over the thirty centuries or so, the hexagrams have taken on a wide range of meaning depending upon the context in which they are applied.

Thus, the use of the *Canon of Changes* as an instrument of reflection and thought is not new, as evidenced by the existence of Taoist, Confucian and Buddhist schools. There have been a few more recent students of the *Chou I*, notably the famous mathematician, Baron Gottfried von Leibniz, one of the inventors of calculus; the psychoanalyst, Carl Jung, a famous student of Sigmund Freud; the Nobelist in literature, Hermann Hesse (author of *The Glass Bead Game*); and others.

Leibniz referred to the *I Ching* as a "Two-Element Arithmetic"; had he lived later he might have viewed it as an example of a Boolean algebra (the foundation of modern computer science).

In particular, the ancient Chinese were farmers, so the hexagrams themselves are shrouded in interpretation as mystical weather-like symbols. Such phenomena, as studied today by meteorologists, are known as the Lorenz Strange Attractors.

Essentially, these are the set of equations which describe turbulence and chaos, the difficulties involved with predicting the weather. The mathematician and philosopher of the Sung Dynasty (A.D. 960-1279), Shao Yung, studied the mythical Fu-hsi's description "following a natural progression of weather conditions". These patterns are depicted as the doubling of two trigrams producing such primitive equations as:

The Kou Hexagram #44, Ch'ien/Sun: The Sky Is Clear and The Wind Comes, traditionally numbered (7,7,7) and (7,9,6):

7 at top	1 1 0	Ch'ien
7 in the fifth	1 1 0	Sky (upper trigram)
7 in the fourth	1 1 0	Sky (upper middle)
7 in the third	1 1 0	Sky (lower middle)
9 in the second	1 1 1	Wind (lower trigram)
6 in the first	0 0 0	Sun

and changing into the T'ung Jen Hexagram #13, Ch'ien/Li: The Wind Brings Heat, traditionally numbered (7,7,7) and (7,8,7):

7 at top	1 1 0	Ch'ien
7 in the fifth	1 1 0	Sky (upper trigram)
7 in the fourth	1 1 0	Sky (upper middle)
7 in the third	1 1 0	Wind (lower middle)
8 in the second	1 1 1	Heat (lower trigram)
7 in the first	0 0 0	Li

Granted, these formulae seem a bit obscure but we must remember that they are "very old".

Thus 110 or (101 or 011) say, can be thought of as a symbolic representation of the static yang numeral (7), (i.e. not the number 7), generated by some means, say flipping three coins at once, (where 1 stands for heads and 0 stands for tails) and 100 (or 001 or 010) the representation of a static yin (8) generated similarly.

If three heads or tails are encountered (9 or 6), the hexagrams are then changing, yielding a distinct pair, as shown above. Note that the above binary symbols do not form a true mathematical description of a binary number in the modern sense, though the ancient scholars may have mysteriously inserted implicit values of 1 or 0 just as in an IEEE-type format which may use an implicit 1 to represent floating point numbers.

Today the *I Ching* is widely used as an oracle as well as a guide to the study of ancient Chinese characters and to the myriad of philosophies inherent in it. It is the gem of Chinese astrology, but has other aspects as well. It has a natural affinity to computer programming, being a Boolean system.

For those who are interested, an unsolved problem, as far as we know, is the generation of the so-called Shchutskii numbers: numbers assigned to the hexagrams concerning the occurrence of the four mantic forms: yuan, heng, li, and chen, curiously extant in exactly half of the 64 hexagrams of the first layer or wing of the text. There seems to be no formulae or patterns as to why they occur in some hexagrams but not in others. Indeed, the *I Ching* has changed much since its inception, and in its incipient stage consisted of oral mantic traditions that lost their original meanings through gradual philological redefinition of the mantic formulae.

The intersection of the host of meanings derived from an inquiry of the *I Ching* brings us onto the frontiers of artificial intelligence. These great varieties of interpretation are employed in certain psychoanalytic games which are user friendly, giving them a sense of volition. For example, the DATA statements in the Hexagram Program can be any statements, phrases or symbols with repetition among the statements. Thus if the computer picks DATA 128 it may be a "Morse Code beep", or an animal noise, or a flashing screen together with a thunder clap followed by some comforting words, and furthermore it may generate one or more DATA statements with such notions.

The Program as Oracle: Seeding RND

There are many ways one may seed the RND function. One way is to write a simple word processor that echoes one's question on the video screen and adds the numerical value of the ASCII string modulo 64 (or something similar)

I will not employ this method but will leave the program at the mercy of arbitrary numeric input by the user to determine a pseudo-random seed. Possibly, by adding in TI$ one may produce a better pseudo-random routine. The theory of random numbers is not a trivial matter and much can be done in this respect.

The DATA statements are chosen according to the formula 126+2*n, where n=0,...,63 as is obvious in the program's DATA listing, but is somewhat more involved as evidenced in the hexagram-naming routine seen in the main body of the program (compare BASIC lines 1 through 42, particularly 25 and 26). I have decoded the appropriate hexagram corresponding to the correct data number. I include the traditional numbering together with the actual name of the hexagram corresponding to that numbering in the DATA statements. Therefore, there are actually two numbers for each hexagram.

To consult the oracle, run the BASIC program, and input any two integers in response to the prompts. The larger the numbers you use, the slower the program will run. That is all there is to it. You play the "Strange Attractor". In deference to the Taoist idea that a hexagram is the time, I include TI$ next to each line.

Of course, you may restructure the whole program (possibly incorporating ideas from the "Animals" program) and open files on a disk governed by the hexagram-naming routine, or do whatever you wish. Even increase the number n to 127 to create heptagrams, or to 255 for octagrams, and so on. You are only limited by your imagination.

In summary, the hexagram-generating program is a computerized *I Ching*. Instead of flipping coins or using yarrow stalks to generate hexagrams and then looking up a hexagram's associated text, everything could be provided in the computer program. This program took a long time to evolve, and many hours of programming and research went into it. We sincerely hope you enjoy it. My special thanks to Prof. Charles Litzinger, Prof. Roy Leipnik, Ingeborg Comstock, James Centtanni, Dr. Ibrahim Mustafa and his wife Truus for their helpful suggestions. All mistakes are my own, though I hope that they are few and far between. Dr. Mustafa and I have written a Text-to-Hexagram Processor in Pascal and Assembly Language. It employs a word processor with onscreen menu and associated files. We would appreciate it if you would drop us a card with your ideas concerning the improvement of the Hexagram program, as well as notes on bugs that you may find.

Please send all correspondence to:

CompuCell
P.O. Box 2493
Goleta, CA 93118

References

1) Blofeld, J., *I Ching*, E.P. Dutton Co. Inc., New York

2) Hesse, H., *The Glass Bead Game*,
Bantam Books Inc., New York

3) Jung, K., *Man and his Symbols*, Dell Publishing Co.

4) Kalgren, B., *Sound and Symbol*, Oxford University Press

5) Legge, J., *I Ching*, Causeway Books, New York

6) Shchutskii, I., *Researches on the I Ching*, Princeton University Press

7) Wilhelm/Baynes, *I Ching*, Princeton University Press

8) Wilhelm, *Eight Lectures on the I Ching*, Princeton University Press

9) Shao Yung, *Plum Blossom Numerology*

10) Wincupp, *Rediscovering the I Ching*, Doubleday, New York

11) *Transactor*, Volume 7, Issue 2

12) *Science News*, 5/86

Listing: The Hexagram Program

(This program will run on all 8-bit Commodore computers).

```
DH 1 rem *** program by audry vilkas and james centanni * copyright 1986 ***
PL 2 printchr$(147):dim w$(64,2):t=0:z=0:b$=chr$(192):b2$=b$+b$:b5$=b2$+b2$+b$
HG 3 input"integer 1";x
LO 4 input"integer 2";y:print:print
CP 5 for x=1 to 6
LL 6 for l=int(rnd(0)*x) to int(rnd(1)*y)
IB 7 i=int(rnd(1)*2)
LB 8 j=int(rnd(1)*2)
OB 9 k=int(rnd(1)*2)
OA 10 next
FA 11 if i=0 and j=0 and k=0 then s=1:p=2:goto 21
KA 12 if i=1 and j=1 and k=1 then s=2:p=1:goto 22
FF 13 s=i+j+k
AB 14 p=s
IN 15 if i=0 and j=0 and k=1 goto 23
JN 16 if i=0 and j=1 and k=0 goto 23
KN 17 if i=1 and j=0 and k=0 goto 23
AO 18 if i=0 and j=1 and k=1 goto 24
BO 19 if i=1 and j=0 and k=1 goto 24
CO 20 if i=1 and j=1 and k=0 goto 24
HJ 21 print"6 ";b2$;" ";b2$;"   ";b5$;"   ";ti$:goto 25 :rem moving yin line
IA 22 print"9 ";b5$;"   ";b2$;" ";b2$;"   ";ti$:goto 25 :rem moving yang line
GB 23 print"8 ";b2$;" ";b2$;"   ";b2$;" ";b2$;"   ";ti$:goto 25 :rem static yin
NK 24 print"7 ";b5$;"   ";b5$;"   ";ti$:goto 25 :rem static yang
BF 25 t=(2^x)*s+t
AH 26 z=(2^x)*p+z
HC 27 next x
KN 29 for x=1 to 64
CA 30 for y=1 to 2
LK 31 read w$(x,y)
JE 32 next y:next x
NL 33 for m=1 to 64
CB 34 if t=val(w$(m,1)) then ty$=w$(m,2)
PO 35 if z=val(w$(m,1)) then z$=w$(m,2)
PA 36 next m
JK 37 print:print:print"data #:"t:print"  hexagram: "ty$:print
NO 38 print"data #:"z:print"  hexagram: "z$
JH 39 print:print:print"again? (y/n)"
MO 40 get a$:if a$="" then 40
GN 41 if a$="y" then run
KC 42 end
PM 126 data 126,  2 receptive*tenth month
IL 128 data 128, 23 splitting apart*ninth month
GA 130 data 130, 30 holding together*third month
LJ 132 data 132, 20 contemplation*fourth or fifth or sixth month
HE 134 data 134, 16 enthusiasm*second month
LM 136 data 136, 35 progress*first month
BF 138 data 138, 45 gathering together*second month (approx. march)
EC 140 data 140, 12 stagnation*seventh month
PM 142 data 142, 15 modesty*eleventh month
FB 144 data 144, 52 keeping still*ninth month (approx. oct.)
IF 146 data 146, 39 obstruction*tenth month (approx. nov.)
GE 148 data 148, 53 gradual development*twelfth month (approx. jan.)
BC 150 data 150, 62 preponderence of the small*twelfth month (approx. jan.)
AO 152 data 152, 56 wanderer*third month (approx. april)
BI 154 data 154, 31 influence*fourth month (approx. may)
MK 156 data 156, 33 retreat*sixth month (approx. july)
BN 158 data 158,  7 army*third month
IM 160 data 160,  4 folly*twelfth month
EA 162 data 162, 29 danger*tenth or eleventh or twelveth month
IB 164 data 164, 59 dispersion*fifth month
OF 166 data 166, 40 deliverance*first month (approx. feb.)
EH 168 data 168, 64 before completion*tenth month
FL 170 data 170, 47 oppression*eighth month
CG 172 data 172,  6 conflict*second month
OL 174 data 174, 46 pushing upwards*eleventh month
EF 176 data 176, 18 work on what is spoiled*second month
LK 178 data 178, 48 the well*fourth month
PJ 180 data 180, 57 gentleness*seventh month
NO 182 data 182, 32 duration*sixth month
BC 184 data 184, 50 the cauldron*fifth month
HC 186 data 186, 62 preponderance of the great*ninth month
FK 188 data 188, 44 coming to meet*fifth month
KM 190 data 190, 24 return*eleventh month
GD 192 data 192, 27 providing nourishment*tenth month
MP 194 data 194,  3 difficulty in the beginning*eleventh month
JE 196 data 196, 42 increase*twelfth month
OF 198 data 198, 51 shock*first or second or third month
AL 200 data 200, 21 biting through*ninth month
HO 202 data 202, 17 following*first month
IN 204 data 204, 25 innocence*eighth month
CA 206 data 206, 36 darkening of the light*eighth month
PH 208 data 208, 22 grace*seventh month
DI 210 data 210, 63 after completion*ninth month
EM 212 data 212, 37 family*fourth month
NH 214 data 214, 55 abundance*fifth month
FM 216 data 216, 30 the clinging*eighth month
MP 218 data 218, 49 revolution*seventh month
GF 220 data 220, 13 fellowship*sixth month
CG 222 data 222, 19 approach*twelfth month
DM 224 data 224, 41 decrease*sixth month
ME 226 data 226, 60 limitation*sixth month
BI 228 data 228, 61 inner truth*tenth month
KF 230 data 230, 54 marrying maiden*eighth month
FO 232 data 232, 38 opposition*eleventh month
CK 234 data 234, 58 joy*seventh or eighth or ninth month
EH 236 data 236, 10 conduct*fifth month
EK 238 data 238, 11 peace*first month
AI 240 data 240, 26 the taming power of the great*seventh month
LO 242 data 242,  5 waiting*first month
JA 244 data 244,  9 the taming power of the small*third month
ON 246 data 246, 34 great power*second month
FI 248 data 248, 14 great possessions*fourth month
NK 250 data 250, 43 breakthrough*third month
DE 252 data 252,  1 the creative*fourth month
```

C64 Hex File Editor

A tool for checking and editing C64 binary files

by Bob Kodadek

After entering a very large machine language listing from a major publication, I was faced with the dilemma of having a seriously flawed program. Knowing that most programs are thoroughly tested prior to publication, the errors were probably mine. Obviously, I had made some serious mistakes in entering the hexadecimal listing, though I did use the checksum utility provided.

Most machine language monitors for the C-64 use an eight byte display line, but some hex program listings do not follow this convention. Since this particular listing used 11 bytes per line, using a machine language monitor to find the errors proved impossible. My only recourse was to write a program that would produce a hardcopy of the object file, identical to the magazine listing, and recheck each byte for error. It took many hours to review and edit the object code until all the errors had been corrected. Why did the checksum program allow these errors?

The answer is that some entry programs produce a "don't care" checksum. It doesn't care about the individual value of a byte of data or its position in the line to be entered. The sum to be checked is produced by adding all the data on a given line to its line number. In BASIC it might look like this:

```
for i=1 to 11
    read b
    ck=ck+b
next
ck=ck+ln
```

While entering a program, if you happen to transpose two or more data bytes, the line is still accepted. For example, if the next two data bytes to be entered were 40 12, you could type them in reverse order, as 12 40. The checksum would never know the difference. It would also be acceptable to enter in correct values if the total sum is still correct. For example, the same two data bytes could be entered as 42 10. The checksum says it's a match, but you and I know otherwise. The result is usually a worthless program. To eliminate this problem we need intelligent checksum programs that care about the data received. There is no magic in producing a checksum program that works, but many publications refuse to bother. Until they do, this is a problem that we must live with. But now the problem is no longer hopeless. There is help available.

The accompanying program, "Hex File Editor", has a function for almost everyone. You may read, write, list, edit, or print the hexadecimal contents of program or sequential files using simple line numbers and a full screen editor. The number of columns displayed is user definable, and access is provided to the disk directory and command channel for easy file maintenance operations. There is a help menu, and commands for converting hex and decimal numbers. It can be used as a fast file copier, to read/alter the load address of a program file, or to convert PRG files to SEQ (or USR) files and vice versa.

The Command Menu

Hex File Editor provides a help menu that displays the available commands, the load address of your file, and its current location in RAM. When operating in the command mode, the program will display the prompt '>', and a blinking cursor. Each command consists of a character and an argument where indicated. Enter the command and press Return. Square brackets show optional arguments, while angle brackets indicate an argument must be specified. After any disk operation, the error channel is read and displayed. The available commands are as follows:

E [line#] - EDIT: This command will display the line specified and enter the full screen editor. All cursor controls function the

same as in the BASIC editor. Press Return to accept the present line and display the next line. You may move the cursor to any line on the screen. To exit this mode type an asterisk or other non-hex character and press Return. Without an argument, editing will start with the first line. Examples:

E 100 Enters edit mode at line 100.
E Enters edit mode at line 1.

L [line#] - LIST: If a line number is not specified, the program will list from beginning to end, otherwise it will list from the specified number. Press Shift to freeze the listing, Ctrl to slow, and Stop to halt. Examples:

L 100 Lists from line 100.
L Lists from line 1.

P [line#] - PRINT: This is the same as list except output also goes to a printer with device number 4. Press Shift to freeze or Stop to exit.

D - DIRECTORY: Displays disk directory. Press the spacebar to stop and start listing. Press Stop to abort.

R - READ FILE: Reads a disk file into memory. You will be asked for the filename. Do not use quotation marks around the filename. Enter no name to abort.

W - WRITE FILE: Writes a file to disk from the current data in memory. If a file already exists, either scratch it or select a new name. You will be asked for the file type and filename.

X - DISK COMMAND Send disk command. All commands are supported. You will be asked for the command. For example, to scratch a file enter **s:filename**.

 # - DEC-TO-HEX: Converts a decimal number (0-65535) to hex. For example, entering **#32768** gives a result of $8000.

$ - HEX-TO-DEC: Converts a hex number to decimal. Leading zeroes are mandatory. For example, **$00FF** will yield 255.

C <#> - COLUMNS: Changes the number of columns displayed. The default display is 8 columns. Only a decimal number from 6 to 11 is accepted.

M - MENU: Use this to return to the command help menu at any time.

Q - EXIT: Exit to BASIC. Performs the equivalent of a cold start, SYS 64738.

Using The Program

Type in, save, and then run the BASIC loader program, listing 1. Hex File Editor is always waiting for the Return key to be pressed. When this occurs in the Edit mode, the screen editor begins to process the line the cursor is placed on. First it reads in the line number and converts it to a two byte binary address in memory. This determines where your data are going to be placed in RAM. Then it converts each pair of screen characters into their binary values, carefully checking for spaces along the way. If it finds an error it prints a question mark at the end of the line, stops all processing, and exits to the command mode. If there is no error, the data are stored in memory.

To check a previously entered program, first use the READ command to place the file into memory and select the proper number of columns for the display line. Do not include any checksum characters in this calculation. You may then list to the screen or printer and recheck each line with its original listing. On very large program listings, this can be done at your leisure. Just mark the listing to show where you left off. Only check the pairs of characters that are the actual machine code in each line of the original listing. The last one or two pairs of characters are usually the checksum.

Mark each line where an error is found. After the entire program has been checked, use the Edit mode to correct the bad lines, then save the program on another disk using the WRITE command. To be on the safe side, don't scratch the original version until you are sure all the bugs are out and you have a working copy.

To Copy Programs Or Files

As a program or sequential file copier, the program uses the RAM area 2048-49152 ($0800-$C000) for storage. This allows for a program length of over 47,000 bytes, about 184 blocks of disk space. To do a copy, perform the READ and WRITE operations from the menu. Unlike other copiers, you only have to read the source file once and can specify a different filename when doing the WRITE. You may then make as many copies as needed, very quickly, by repeating the WRITE command.

To convert a program file to a sequential file, or vice versa, just make a copy. When asked for the file type, enter 'P' (PRG), 'S' (SEQ), or 'U' (USR).

Changing The Load Address

When listing program files, the first two bytes in line number one will be the load address in low-byte, high-byte format. By changing this address and writing a new file, you can relocate a program that uses the ,8,1 syntax. This can be used on sprite data, hi-res screens, or relocatable machine language programs. First read in the file and use the DEC-TO-HEX command to calculate a two byte hex address. Use the EDIT command to alter the two bytes and then save the new file using WRITE.

Remember, when referring to a hex address such as $0800 (2048), the first two characters represent the high byte and the last two are the low byte. In 6502-6510 machine code an address will appear low byte first. In other words the two byte load address in the above example would appear in line number one as 00 (low byte) and then 08 (high byte).

Listing 1: Hexed.gen

```
HH  10 rem c64 hex file editor
EO  20 rem (c) 1987 bob kodadek
JH  30 rem     3164 surrey lane
IF  40 rem     aston, pa  19014
AF  50 rem
JJ  60 ml=49152: print "reading..."
LF  70 for i=0 to 1510
GE  80 read by: poke ml+i,by: ck=ck+by
OF  90 next
FE  100 if ck<>180036 then print "data error!": end
BM  110 sys ml
MO  120 :
CJ  1000 data  32, 238, 196, 162,   0, 134, 251, 132
OP  1010 data 252, 142,   0,   8, 142,   1,   8, 142
IP  1020 data 134,   2, 169,  15, 141,  32, 208, 141
FF  1030 data  33, 208, 169, 147,  32, 210, 255, 162
OB  1040 data   2, 160,  12,  24,  32, 240, 255,  32
FJ  1050 data 255, 195,  72,  69,  88,  32,  70,  73
ML  1060 data  76,  69,  32,  69,  68,  73,  84,  79
IF  1070 data  82,  13,  13,  13,  32,  32,  77,  69
PH  1080 data  78,  85,  32,  32,  32,  32,  40,  67
CJ  1090 data  41,  32,  49,  57,  56,  55,  32,  66
GP  1100 data  79,  66,  32,  75,  79,  68,  65,  68
KN  1110 data  69,  75,  13,  13,  69,  45,  69,  68
EN  1120 data  73,  84,  13,  76,  45,  76,  73,  83
HL  1130 data  84,  13,  80,  45,  80,  82,  73,  78
GO  1140 data  84,  13,  68,  45,  68,  73,  82,  69
JA  1150 data  67,  84,  79,  82,  89,  13,  82,  45
LO  1160 data  82,  69,  65,  68,  32,  70,  73,  76
HB  1170 data  69,  13,  87,  45,  87,  82,  73,  84
FC  1180 data  69,  32,  70,  73,  76,  69,  13,  88
GC  1190 data  45,  68,  73,  83,  75,  32,  67,  79
FD  1200 data  77,  77,  65,  78,  68,  13,  35,  45
JE  1210 data  68,  69,  67,  32,  84,  79,  32,  72
NE  1220 data  69,  88,  13,  36,  45,  72,  69,  88
ED  1230 data  32,  84,  79,  32,  68,  69,  67,  13
JH  1240 data  67,  45,  67,  79,  76,  85,  77,  78
NG  1250 data  83,  13,  77,  45,  77,  69,  78,  85
KA  1260 data  13,  81,  45,  69,  88,  73,  84,   0
FC  1270 data 162,  18, 160,  20,  24,  32, 240, 255
PJ  1280 data  32, 255, 195,  76,  79,  65,  68,  32
DK  1290 data  65,  68,  68,  82,  69,  83,  83,  58
HK  1300 data  36,   0, 174,   0,   8, 173,   1,   8
KP  1310 data  32,  82, 196, 162,  19, 160,  20,  24
OG  1320 data  32, 240, 255,  32, 255, 195,  79,  66
ML  1330 data  74,  69,  67,  84,  58,  36,   0, 162
EM  1340 data   0, 169,   8,  32,  82, 196,  32, 255
MK  1350 data 195,  45,  36,   0, 166, 251, 165, 252
FN  1360 data  32,  82, 196,  32, 179, 197, 169, 240
NE  1370 data 133, 130, 169, 239, 133, 131, 169,  13
LJ  1380 data  32, 210, 255, 162,  38, 164, 211, 169
CK  1390 data  32, 145, 209, 200, 202, 208, 250, 169
DH  1400 data  62,  32, 210, 255,  32, 247, 196,  32
LE  1410 data 115,   0, 217,  99, 193, 240,   8, 200
DE  1420 data 192,  12, 208, 246,  76,  35, 193, 152
MO  1430 data  10, 170, 189, 112, 193,  72, 189, 111
MK  1440 data 193,  72,  96,  82,  87,  76,  69,  88
BI  1450 data  68,  77,  81,  80,  35,  36,  67, 135
ED  1460 data 193,  11, 194, 141, 194,  99, 195, 163
BH  1470 data 194, 239, 194,  17, 192, 206, 194,  97
FC  1480 data 194, 217, 195, 202, 195, 235, 195,   0
LH  1490 data  32, 201, 196,  32, 179, 196,  32, 238
KL  1500 data 196, 162,   3,  32, 198, 255, 160,   0
FF  1510 data  32, 207, 255, 145, 253,  32,   5, 194
HG  1520 data  32, 183, 255, 141, 204, 197, 201,  64
IB  1530 data 240,   8, 173, 204, 197, 208,  21,  76
OD  1540 data 152, 193, 145, 253, 152, 200, 145, 253
NB  1550 data 192,   3, 208, 249, 165, 253, 133, 251
LH  1560 data 165, 254, 133, 252,  32, 179, 197, 165
JH  1570 data 186,  32, 180, 255, 169, 111, 133, 185
NM  1580 data  32, 150, 255, 169,  13,  32, 210, 255
IC  1590 data  32, 165, 255, 201,  13, 240,   6,  32
DI  1600 data 210, 255,  76, 216, 193,  32, 210, 255
GN  1610 data  32, 171, 255,  32, 244, 193,  32, 207
OJ  1620 data 255,  76,  18, 192,  32, 255, 195,  80
HN  1630 data  82,  69,  83,  83,  32,  82,  69,  84
OJ  1640 data  85,  82,  78,   0,  96, 230, 253, 208
AH  1650 data   2, 230, 254,  96,  32, 255, 195,  84
NN  1660 data  89,  80,  69,  32,  40,  80,  47,  83
PB  1670 data  47,  85,  41,  58,   0,  32, 247, 196
GJ  1680 data  32, 115,   0, 141, 229, 197,  32, 201
GN  1690 data 196, 162,   3, 189, 227, 197, 153,   0
KG  1700 data   2, 200, 202,  16, 246, 132, 183,  32
JN  1710 data 179, 196, 162,   3,  32, 201, 255,  32
AH  1720 data 238, 196, 169,  54, 133,   1, 165, 253
HP  1730 data 197, 251, 208,   6, 165, 254, 197, 252
MO  1740 data 240,  13, 160,   0, 177, 253,  32, 168
LE  1750 data 255,  32,   5, 194,  76,  70, 194,  76
MI  1760 data 196, 193, 169,   4, 133, 186,  32, 177
EG  1770 data 255, 169,  96, 133, 185,  32, 147, 255
BI  1780 data  32, 255, 195,  13,  82,  69,  65,  68
OD  1790 data  89,  63,  32,   0,  32, 244, 193, 169
FB  1800 data  13,  32, 172, 196,  32, 168, 197,  32
IE  1810 data 228, 255, 201,  13, 208, 246,  32,   1
MP  1820 data 197,  32, 168, 197,  32,  29, 196, 173
EA  1830 data 141,   2, 201,   1, 240, 249,  32, 109
IB  1840 data 196,  76, 145, 194,  32, 255, 195,  67
EK  1850 data  79,  77,  77,  65,  78,  68,  58,   0
NN  1860 data  32, 247, 196, 165, 186,  32, 177, 255
EH  1870 data 169, 111, 133, 185,  32, 147, 255, 160
FD  1880 data   0, 185,   0,   2, 240,   6,  32, 168
BN  1890 data 255, 200, 208, 245,  76, 196, 193,  32
LP  1900 data 255, 195,  13,  69,  88,  73,  84,  63
DI  1910 data  32,  40,  89,  47,  78,  41,   0,  32
AN  1920 data 228, 255, 201,  78, 240,   7, 201,  89
NE  1930 data 208, 245,  76, 226, 252,  76,  18, 192
GL  1940 data 169,   1, 162,  99, 160, 195,  32, 189
PP  1950 data 255, 169,  96, 133, 185,  32, 213, 243
HD  1960 data 165, 186,  32, 180, 255, 165, 185,  32
PK  1970 data 150, 255, 169,   0, 133, 144, 160,   3
JD  1980 data 132, 183,  32, 165, 255, 133, 195,  32
MN  1990 data 165, 255, 133, 196, 164, 144, 208,  61
KC  2000 data 164, 183, 136, 208, 235, 166, 195, 165
EL  2010 data 196,  32, 205, 189, 169,  32,  32, 210
CI  2020 data 255,  32, 165, 255, 166, 144, 208,  37
LJ  2030 data 201,   0, 240,  24,  32, 210, 255,  32
DB  2040 data 225, 255, 240,  25,  32, 228, 255, 240
LD  2050 data 232, 201,  32, 208, 228,  32, 228, 255
HB  2060 data 240, 251, 208, 221, 169,  13,  32, 210
NE  2070 data 255, 160,   2, 208, 179,  32,  66, 246
MG  2080 data  76,  35, 193,  36,  32,   1, 197, 169
KD  2090 data  13,  32, 210, 255,  32,  29, 196, 160
NN  2100 data  34, 169, 157,  32, 210, 255, 136,  16
LG  2110 data 248, 169,   0, 168,  32,  93, 241, 201
AL  2120 data  13, 240,   6, 153,   0,   2, 200, 208
GF  2130 data 243, 162, 255, 160,   1, 134, 122, 132
JE  2140 data 123,  32,   1, 197,  32, 121,   0, 201
OK  2150 data  45, 208,  21, 160,   0,  32,  59, 197
DN  2160 data 153,   0,   1,  32, 115,   0, 200, 204
FI  2170 data 205, 197, 240,  12, 201,  32, 240, 237
FL  2180 data 169,  63,  32, 210, 255,  76,  35, 193
BH  2190 data 160,   0, 185,   0,   1, 145, 253, 200
GL  2200 data 204, 205, 197, 208, 245,  32, 109, 196
```

```
DP  2210 data  76, 103, 195,  32,  65, 197, 168,  32
NH  2220 data  65, 197, 170, 152,  32, 205, 189,  76
GP  2230 data  35, 193,  32, 114, 197, 169,  36,  32
CP  2240 data 210, 255, 166,  20, 165,  21,  32,  82
OG  2250 data 196,  76,  35, 193,  32, 114, 197, 165
JO  2260 data  20, 201,   6, 144,   7, 201,  12, 176
LO  2270 data   3, 141, 205, 197,  76,  35, 193, 104
JN  2280 data 133,  34, 104, 133,  35, 208,   3,  32
PK  2290 data 210, 255, 160,   0, 230,  34, 208,   2
GJ  2300 data 230,  35, 177,  34, 208, 241, 165,  35
PE  2310 data  72, 165,  34,  72,  96,  32, 151, 196
ON  2320 data 160,   0, 185, 206, 197, 240,   6,  32
JG  2330 data 172, 196, 200, 208, 245, 169,  45,  32
LE  2340 data 172, 196, 160,   0, 140, 203, 197, 177
HH  2350 data 253,  32,  86, 196, 169,  32,  32, 172
GI  2360 data 196, 238, 203, 197, 172, 203, 197, 204
OG  2370 data 205, 197, 208, 235, 169,  13,  32, 172
NI  2380 data 196,  96,  32,  86, 196, 138,  72,  74
NN  2390 data  74,  74,  74,  32,  97, 196, 104,  41
JB  2400 data  15,   9,  48, 201,  58, 144,   2, 105
FJ  2410 data   6,  32, 172, 196,  96, 162,   3, 254
AL  2420 data 206, 197, 189, 206, 197, 201,  58, 208
IJ  2430 data   8, 169,  48, 157, 206, 197, 202,  16
OJ  2440 data 238, 238, 201, 197, 208,   3, 238, 202
PN  2450 data 197, 165, 253, 109, 205, 197, 133, 253
LK  2460 data 165, 254, 105,   0, 133, 254,  96,  56
EN  2470 data 165, 253, 229, 251, 133,  81, 165, 254
CN  2480 data 229, 252,   5,  81, 176,   1,  96, 104
LN  2490 data 104,  76,  35, 193,  32, 168, 255,  32
OI  2500 data 210, 255,  96, 165, 183, 162,   0, 160
CG  2510 data   2,  32, 189, 255, 169,   3, 162,   8
CA  2520 data 160,   3,  32, 186, 255,  32, 192, 255
CG  2530 data  96,  32, 255, 195,  70,  73,  76,  69
BD  2540 data  32,  78,  65,  77,  69,  58,   0,  32
JB  2550 data 247, 196, 185,   0,   2, 240,   3, 200
DE  2560 data 208, 248, 132, 183, 164, 183, 208,   5
FO  2570 data 104, 104,  76,  18, 192,  96, 162,   0
JB  2580 data 160,   8, 134, 253, 132, 254,  96,  32
JO  2590 data  96, 165, 134, 122, 132, 123, 160,   0
CD  2600 data  96, 169,   1, 141, 201, 197, 169,   0
FN  2610 data 141, 202, 197,  32, 238, 196, 169,  48
FK  2620 data 160,   3, 153, 206, 197, 136,  16, 250
GL  2630 data 238, 209, 197,  32, 114, 197, 165,  20
ND  2640 data 208,   4, 165,  21, 240,  20, 173, 201
AB  2650 data 197, 197,  20, 208,   7, 173, 202, 197
OI  2660 data 197,  21, 240,   6,  32, 109, 196,  76
FL  2670 data  38, 197,  96, 169, 234, 133, 130, 133
IK  2680 data 131,  32, 115,   0,  32,  95, 197, 142
GO  2690 data 199, 197,  32, 115,   0,  32,  95, 197
MN  2700 data 142, 200, 197, 173, 199, 197,  10,  10
PJ  2710 data  10,  10,  24, 109, 200, 197,  96, 162
KG  2720 data   0, 221, 211, 197, 240, 248, 232, 224
DB  2730 data  16, 208, 246, 104, 104, 104, 104,  76
FM  2740 data 176, 195, 162,   0, 134,  20, 134,  21
HM  2750 data  32, 115,   0, 176,  42, 233,  47, 133
MM  2760 data   7, 165,  21, 133,  34, 165,  20,  10
LO  2770 data  38,  34,  10,  38,  34, 101,  20, 133
KN  2780 data  20, 165,  34, 101,  21, 133,  21,   6
NN  2790 data  20,  38,  21, 165,  20, 101,   7, 133
NG  2800 data  20, 144, 213, 230,  21, 208, 209,  96
KE  2810 data  32, 225, 255, 208,   5, 104, 104,  76
JD  2820 data  35, 193,  96,  32, 174, 255,  32, 204
DC  2830 data 255, 165, 184,  32, 195, 255, 169,   8
PH  2840 data 133, 186, 169,  55, 133,   1,  96,   0
AE  2850 data   0,   0,   0,   0,   0,   8,  48,  48
KG  2860 data  48,  48,   0,  48,  49,  50,  51,  52
FK  2870 data  53,  54,  55,  56,  57,  65,  66,  67
JC  2880 data  68,  69,  70,  87,  44,  80,  44
```

Listing 2: Hexed.src

```
*------------------------------*
*  c64 hex  file editor        *
*  (c) 1987 bob kodadek        *
*       3164 surrey lane       *
*       aston, pa  19014       *
*------------------------------*
*  merlin-128 macro-assembler  *

chrget   = $73              ;character get routine
chrgot   = $79              ;get character again
endadr   = $fb              ;pointer:highest address
ramptr   = $fd              ;pointer:to ram
temp     = $51              ;temporary usage
flen     = $b7              ;filename length
buffer   = $0100            ;work area
buf      = $0200            ;system input buffer
sa       = $0800            ;start of usable ram
second   = $ff93            ;kernal equates
tksa     = $ff96
acptr    = $ffa5
ciout    = $ffa8
untlk    = $ffab
unlsn    = $ffae
listen   = $ffb1
talk     = $ffb4
readst   = $ffb7
setlfs   = $ffba
setnam   = $ffbd
open     = $ffc0
close    = $ffc3
chkin    = $ffc6
chkout   = $ffc9
clrchn   = $ffcc
chrin    = $ffcf
chrout   = $ffd2
stop     = $ffe1
getin    = $ffe4
plot     = $fff0

* note: labels beginning with "]" are variables
*       and are used for backward branching only.

         org $c000

start    jsr setpnt         ;set ramptr
         ldx #0             ;zero end address
         stx endadr
         sty endadr+1
         stx sa             ;zero load address
         stx sa+1
         stx $0286          ;black for text color
help     lda #15            ;color code for grey
         sta $d020          ;set border color
         sta $d021          ;set background color
         lda #$93           ;clr screen
         jsr chrout
         ldx #2             ;locate cursor
         ldy #12
         clc
         jsr plot           ;print menu screen
         jsr primm
         txt 'hex file editor'0d0d0d
         txt '  menu    (c) 1987 bob kodadek'0d0d
         txt 'e-edit'0d
         txt 'l-list'0d
         txt 'p-print'0d
         txt 'd-directory'0d
         txt 'r-read file'0d
         txt 'w-write file'0d
         txt 'x-disk command'0d
         txt '#-dec to hex'0d
         txt '$-hex to dec'0d
         txt 'c-columns'0d
         txt 'm-menu'0d
         txt 'q-exit'00
```

```
        ldx #18             ;locate cursor
        ldy #20
        clc
        jsr plot
        jsr primm
        txt 'load address:$'00
        ldx sa
        lda sa+1
        jsr printhex        ;print load address
        ldx #19             ;locate cursor
        ldy #20
        clc
        jsr plot
        jsr primm
        txt 'object:$'00
        ldx #<sa
        lda #>sa
        jsr printhex        ;print start address
        jsr primm
        txt '-$'00
        ldx endadr
        lda endadr+1
        jsr printhex        ;print ending address
* get command and execute
getcom  jsr restore         ;restore channels
        lda #$f0            ;chrget restored always
        sta $82
        lda #$ef
        sta $83
        lda #$0d
        jsr chrout          ;print a cr
        ldx #38             ;erase command line
        ldy $d3
        lda #$20
]erase  sta ($d1),y
        iny
        dex
        bne ]erase
        lda #'>'
        jsr chrout          ;print command prompt
        jsr input           ;get input
        jsr chrget          ;read character
]loop   cmp table,y         ;compare to command table
        beq docom           ;if found, do it
        iny                 ;else, get more
        cpy #$0c            ;tested all?
        bne ]loop
        jmp getcom          ;not legal command
docom   tya                 ;get index into a
        asl                 ;mulitply x 2
        tax
        lda adr+1,x         ;look up address
        pha                 ;push it on stack
        lda adr,x
        pha
        rts                 ;jump to command routine

table   asc 'rwlexdmqp#$c'
adr     da  read-1          ;read file
        da  write-1         ;write file
        da  list-1          ;list to screen
        da  edit-1          ;full screen editor
        da  diskc-1         ;send disk command
        da  direct-1        ;read directory
        da  help-1          ;produce main menu
        da  quit-1          ;return to basic
        da  plist-1         ;list to printer
        da  dechex-1        ;convert decimal to hex
        da  hexdec-1        ;convert hex to decimal
        da  change-1        ;select number of columns
        hex 00

*** command routines ***

read = *
* reads prg, seq, or usr file into ram
        jsr fname           ;input filename
        jsr setlog          ;set logical file
        jsr setpnt          ;point to $0800
        ldx #$03
        jsr chkin           ;open input channel
        ldy #$00
]loop   jsr chrin           ;input character
        sta (ramptr),y  ;store in ram
        jsr incpnt          ;increment ramptr
        jsr readst          ;read status byte
        sta erbyt           ;save it
        cmp #64             ;test for eol
        beq eof
        lda erbyt           ;test for error
        bne rderr           ;read error channel
        jmp ]loop
eof     sta (ramptr),y      ;store eof marker ($40)
        tya                 ;y=0
]loop   iny
        sta (ramptr),y  ;and a few zero bytes
        cpy #3
        bne ]loop
        lda ramptr          ;move ramptr to endadr
        sta endadr
        lda ramptr+1
        sta endadr+1
rderr   jsr restore         ;clear channels
        lda $ba             ;read error channel
        jsr talk            ;device 8 talks
        lda #$6f            ;from command channel
        sta $b9
        jsr tksa
        lda #$0d
        jsr chrout          ;print a cr
]get    jsr acptr           ;input serial byte
        cmp #$0d            ;test for cr
        beq enderr
        jsr chrout          ;print the byte
        jmp ]get
enderr  jsr chrout          ;print the cr
        jsr untlk           ;stop talking
        jsr prrt            ;prompt "press return"
        jsr chrin           ;wait for <return>
        jmp help            ;display menu
prrt    jsr primm
        txt 'press return'00
        rts

incpnt  inc ramptr          ;increment ram pointer
        bne r1
        inc ramptr+1
r1       rts

write = *
* writes a binary file in prg, seq, or usr format
        jsr primm
        txt 'type (p/s/u):'00
]get    jsr input           ;get user's file type
        jsr chrget
        sta ftyp            ;save it
        jsr fname           ;get filename
        ldx #$03
]loop   lda wr,x
        sta buf,y           ;append file type
        iny
        dex
        bpl ]loop
        sty flen            ;set file length
        jsr setlog          ;set up logical file
        ldx #$03
        jsr chkout          ;open output channel
        jsr setpnt          ;point to start ($0800)
        lda #$36            ;basic roms out
        sta $01
]loop   lda ramptr          ;test for end
        cmp endadr
        bne w1
        lda ramptr+1
```

```
        cmp endadr+1
        beq w2
w1      ldy #$00
        lda (ramptr),y  ;get ram byte
        jsr ciout       ;output byte
        jsr incpnt      ;increment ramptr
        jmp ]loop
w2      jmp rderr       ;read error channel

plist = *
* lists to screen and printer
        lda #4          ;set device to #4
        sta $ba
        jsr listen      ;printer listens
        lda #$60
        sta $b9
        jsr second      ;secondary address 0
        jsr primm       ;prompt "ready?"
        dfb 13
        txt 'ready? '00
        jsr prrt        ;prompt "press return"
        lda #$0d
        jsr senchr      ;print a cr
]loop   jsr ckstop      ;test stop key
        jsr $ffe4       ;wait for <return>
        cmp #$0d
        bne ]loop

list = *
* list to screen and a listener, if present
        jsr calcln      ;calculate line number
]loop   jsr ckstop      ;check stop key
        jsr line        ;output line
wait    lda 653         ;check shift key status
        cmp #$01        ;freeze listing if active
        beq wait
        jsr incln       ;increment line number
        jmp ]loop       ;do more

diskc = *
* sends user command to drive
        jsr primm       ;prompt user
        txt 'command:'00
        jsr input       ;get command
        lda $ba         ;device 8 listens
        jsr listen
        lda #$6f        ;command channel
        sta $b9
        jsr second
        ldy #$00
]loop   lda buf,y       ;read input buffer
        beq d1
        jsr ciout       ;send command string
        iny
        bne ]loop
d1      jmp rderr       ;read error channel

quit = *
* routine to exit back to basic
        jsr primm       ;prompt
        db  13
        txt 'exit? (y/n)'00
]loop   jsr getin       ;getkey
        cmp #'n'
        beq q1
        cmp #'y'
        bne ]loop
        jmp 64738       ;system reset
q1      jmp help        ;or return to menu

direct = *
* displays directory from current drive
        lda #$01        ;setup filename "$"
        ldx #<file
        ldy #>file
        jsr setnam
        lda #$60
        sta $b9
        jsr $f3d5
        lda $ba         ;device 8 talks
        jsr talk
        lda $b9
        jsr tksa
        lda #$00        ;clear status byte
        sta $90
        ldy #$03
get     sty $b7
        jsr acptr       ;get # blocks in file
        sta $c3         ;save low byte
        jsr acptr
        sta $c4         ;and high byte
        ldy $90         ;test status byte
        bne dout        ;if not 0 then exit
        ldy $b7
        dey
        bne get
        ldx $c3
        lda $c4
        jsr $bdcd       ;print # blocks
        lda #$20        ;print a space
        jsr chrout
]loop   jsr acptr       ;read filename
        ldx $90
        bne dout        ;test status
        cmp #$00        ;test for new line
        beq newln
        jsr chrout      ;print a character
        jsr stop        ;test stop key
        beq dout
        jsr getin       ;get keypress
        beq ]loop
        cmp #$20        ;test for space bar
        bne ]loop       ;freeze if pressed
]wait   jsr getin       ;wait for keypress
        beq ]wait
        bne ]loop       ;continue listing
newln   lda #$0d        ;print cr
        jsr chrout
        ldy #$02
        bne get         ;go do more
dout    jsr $f642       ;restore channels
        jmp getcom      ;next command
file    asc '$'

edit = *
* full screen editor routine
        jsr calcln      ;calculate line number
e1      lda #$0d        ;print a cr
        jsr chrout
        jsr line        ;print the line
        ldy #34
]loop1  lda #157        ;reposition cursor
        jsr chrout
        dey
        bpl ]loop1
        lda #$00        ;input screen line
        tay
]get    jsr $f15d
        cmp #$0d
        beq g1
        sta buf,y       ;save in buffer
        iny
        bne ]get
g1      ldx #$ff        ;point chrget to buf
        ldy #$01
        stx $7a
        sty $7b
        jsr calcln      ;read line number
        jsr chrgot      ;get last char read
        cmp #'-'        ;compare to "-"
        bne nothex      ;exit if not
        ldy #$00
]loop2  jsr edhex       ;editor hex asc to binary
```

```
        sta buffer,y        ;save binary
        jsr chrget          ;get next character
        iny
        cpy col             ;last column?
        beq skip            ;then skip test
        cmp #$20            ;else, must be a space
        beq ]loop2
nothex  lda #'?'            ;data error! exit
        jsr chrout
        jmp getcom
* if no errors, now store the data in ram
skip    ldy #0
]loop3  lda buffer,y        ;get binary value
        sta (ramptr),y  ;store it in ram
        iny
        cpy col             ;done all columns?
        bne ]loop3
        jsr incln           ;increment line number
        jmp el

hexdec = *
* outputs decimal from ascii hexadecimal input
        jsr rdhex           ;get high byte
        tay                 ;save it
        jsr rdhex           ;get low byte
        tax
        tya
        jsr $bdcd           ;print in decimal
        jmp getcom          ;next command

dechex = *
* outputs hexadecimal number from ascii decimal input
        jsr ascint          ;asc to integer
        lda #'$'            ;print "$"
        jsr chrout
        ldx $14             ;get low byte integer
        lda $15             ;then high byte
        jsr printhex        ;output number in hex
        jmp getcom          ;get next command

change = *
* user sets number of columns displayed (6-11)
        jsr ascint          ;asc to integer
        lda $14             ;get low byte
        cmp #$06            ;<6 columns?
        bcc ch1             ;then exit
        cmp #12             ;=>12 columns?
        bcs ch1             ;also exit
        sta col             ;store if 6-11
ch1     jmp getcom          ;get next command

*************** subroutines ***************

* c-64 print immediate routine allows imbedded string
primm   pla                 ;remove return address
        sta $22             ;save it as current pc
        pla
        sta $23
        bne nxtchar         ;branch always
pchr    jsr chrout
nxtchar ldy #0
        inc $22             ;increment position
        bne nc
        inc $23
nc      lda ($22),y         ;get text
        bne pchr            ;print until #$00
        lda $23             ;new return address
        pha
        lda $22
        pha
        rts                 ;get next instruction

* outputs line to current channel or listener
line    jsr tstend          ;test for last line
        ldy #$00
]loop   lda linum,y
        beq 11
        jsr senchr          ;send line number
        iny
        bne ]loop
11      lda #'-'
        jsr senchr
        ldy #0
        sty cnt             ;zero counter
]jloop  lda (ramptr),y      ;read binary in ram
        jsr prbyte          ;output as hex
        lda #$20
        jsr senchr          ;output space
        inc cnt             ;increment counter
        ldy cnt             ;compare to #columns
        cpy col
        bne ]jloop
        lda #$0d
        jsr senchr          ;output cr
        rts

* output hex string from two byte integer
printhex jsr prbyte         ;print a reg
        txa                 ;print x reg
prbyte  pha                 ;save a
        lsr                 ;shift high nibble down
        lsr
        lsr
        lsr
        jsr phex            ;print it
        pla                 ;pull original byte
prnib   and #$0f            ;mask low nibble
phex    ora #$30
        cmp #$3a            ;decimal digit?
        blt jco             ;branch if so
        adc #6              ;add offset for hex
jco     jsr senchr
        rts

incln   ldx #$03            ;increments line number
]loop   inc linum,x
        lda linum,x
        cmp #$3a
        bne in1
        lda #$30
        sta linum,x
        dex
        bpl ]loop
in1     inc lnum
        bne in2
        inc lnum+1
in2     lda ramptr          ;update ram pointer
        adc col
        sta ramptr
        lda ramptr+1
        adc #$00
        sta ramptr+1
        rts

tstend  sec                 ;test for highest address
        lda ramptr          ;double-byte comparison
        sbc endadr
        sta temp
        lda ramptr+1
        sbc endadr+1
        ora temp
        bcs ts1             ;stop if greater
        rts                 ;else ok
ts1     pla
        pla
        jmp getcom

senchr  jsr ciout           ;send byte to listener
        jsr chrout          ;send byte to screen
        rts

setlog  lda flen            ;open logical file
        ldx #$00
        ldy #$02
        jsr setnam
```

```
        lda #$03
        ldx #$08
        ldy #$03
        jsr setlfs
        jsr open
        rts

fname   jsr primm           ;prompt
        txt 'file name:'00
        jsr input           ;get file name
]loop   lda buf,y           ;get length
        beq out
        iny
        bne ]loop
out     sty flen            ;save it
        ldy flen            ;test for no file name
        bne f1
        pla
        pla
        jmp help
f1      rts

setpnt  ldx #<sa            ;reset ram pointer
        ldy #>sa
        stx ramptr
        sty ramptr+1
        rts

input   jsr $a560           ;get user input
        stx $7a             ;point chrget
        sty $7b
        ldy #0
        rts

* this routine translates an ascii line number into
* the needed location in ram and sets the pointer
* (ramptr) accordingly through incln.
calcln  lda #$01            ;set integer line# to 1
        sta lnum
        lda #$00
        sta lnum+1
        jsr setpnt          ;set ramptr to $0800
        lda #$30            ;set asc line# to 0001
        ldy #$03
]loop   sta linum,y
        dey
        bpl ]loop
        inc linum+3
        jsr ascint          ;get integer
        lda $14             ;greater than 0?
        bne cal1            ;yes, continue
        lda $15             ;else, exit
        beq cal3
cal1    lda lnum            ;get line number
        cmp $14             ;test low byte
        bne cal2            ;same?
        lda lnum+1
        cmp $15             ;test high byte
        beq cal3            ;same?
cal2    jsr incln           ;increment # & build string
        jmp cal1
cal3    rts

* this routine translates ascii hex into binary. the
* first entry point modifies the chrget routine to
* accept space characters for the screen editor.
edhex   lda #$ea            ;modify chrget for edit
        sta $82             ;store two nop instr.
        sta $83
rdhex   jsr chrget          ;get ascii character
        jsr tsthex          ;test for hex
        stx hex1            ;store it
        jsr chrget          ;get next char.
        jsr tsthex          ;test for hex
        stx hex1+1          ;store it
        lda hex1            ;get first value
        asl                 ;multiply by 16
        asl
        asl
        asl
        clc                 ;add second value
        adc hex1+1          ;now we have binary
gothex  rts

tsthex  ldx #$00            ;test for hex 0-f
]loop   cmp hex,x           ;compare to table
        beq gothex          ;x reg returns 0-15
        inx                 ;else, do more
        cpx #$10            ;any more left?
        bne ]loop           ;not found
        pla
        pla                 ;pull 2 addresses
        pla
        pla
        jmp nothex          ;report error!

* this routine converts ascii into a two-byte integer
* as in the basic rom routine, but handles 0-65535.
ascint  ldx #$00
        stx $14
        stx $15
]loop   jsr chrget          ;get asc character
        bcs as1             ;set when not asc numeric
        sbc #$2f            ;includes carry
        sta $07             ;save remainder (0-9)
        lda $15             ;build two byte integer
        sta $22             ;temp area
        lda $14
        asl
        rol $22
        asl
        rol $22
        adc $14
        sta $14
        lda $22
        adc $15
        sta $15
        asl $14
        rol $15
        lda $14
        adc $07
        sta $14
        bcc ]loop
        inc $15
        bne ]loop
as1     rts

ckstop  jsr stop            ;stop key pressed?
        bne nostop          ;if not, then continue
        pla                 ;else remove return
        pla                 ;address from stack
        jmp getcom          ;and jump to command
nostop  rts                 ;routine.

restore jsr unlsn           ;unlisten device
        jsr clrchn          ;clear channels
        lda $b8             ;get file number
        jsr close           ;close logical file
        lda #$08            ;set device to 8
        sta $ba
        lda #$37            ;basic rom's in
        sta $01
        rts

hex1    ds  2               ;storage
lnum    ds  2               ;holds line number
cnt     ds  1               ;counter
erbyt   ds  1               ;holds status byte
col     db  8               ;holds # columns
linum   asc '0000'00        ;asc line #
hex     asc '0123456789'
        asc 'abcdef'
wr      asc 'w,'            ;file direction
ftyp    asc 'p,'            ;file type
```

On the C Side...

A C follow-up, and some REU notes

by Adrian Pepper

Adrian actually sent this article to us as a letter, but rather than risk it possibly being overlooked in the midst of the Letters section, we have decided to present it in this form. The programs he refers to near the end of the article will be included on the ***Transactor*** *disk for this issue.*

I have been an avid reader of *Transactor* for several years, and find it the most informative magazine around for serious users of Commodore computers. I was especially glad to see two articles directly relating to the user of the Power C compiler and environment in Volume 8, Issue 5. Unless playing a game, my Commodore 64 spends most of its time running in this environment. (I may be a serious user, but I'm not dour.)

The explanation of the object file format used by Power C was especially good. Both articles, however, show what a lack of communication there can be in the hobbyist computing field.

Library Maintenance and Compatible Assembly

First, "Maintaining the Power C Library", by Eric Giguere, while a very precise, well written article, actually describes a BASIC program that duplicates the functionality of an existing C program, "lib.c", which is included in source form on the Power C distribution disk! It can easily be compiled and used within the Power C Shell, obviating the need for flipping between the Power C and BASIC environments. Perhaps for some readers, though, the BASIC version is more meaningful. And, although I like C, I must admit a useful C program does not fill pages as efficiently as the equivalent BASIC.

Second, David Godshall's "The Link Between C and Assembly" verifies and documents in one place several things I had run across before, and fills in a few details I wasn't sure about. But David's wish has been granted! For well over a year now, C/ASSM Revision 2.0 has been available. It's a public domain C program, from Mark R. Rinfret of Portsmouth, RI, and Ray L. Zarling, of Turlock, CA, that was derived from a PD generic 6502 assembler contributed to USENET by J.H. van Ornum of AT&T Bell Laboratories. Mark Rinfret and Ray Zarling added the necessary "back-end" to generate Power C object files.

The progam and source are available on the Pro-Line Power C BBS, as "cassm.arc". Another program, "ra", for "Reverse Assembler", is also available there; it translates Power C object files into source (almost) suitable for this assembler.

Now, however, Spinnaker (who market Power C) are selling a different assembler package, Power Assembler, which is very good and produces Power C compatible object files. It is as reasonably priced as Power C itself ($40-60 Cdn. in Toronto).

Character Promotion

There is also at least one technical error in David's article. Many people do not understand that expressions in the C language involving character variables (*chars*) are actually integer expressions. When a *char* is used in an expression, it is implicitly "promoted" (lengthened) to an integer [1]. All parameters in the parameter list for a function call are expressions. Therefore, a call to a function using a *char* as a parameter will actually pass the equivalent *int* (with a zero high byte in Power C, a signed extension in most other implementations [2]). Therefore in the sample call given, FRED's Age would actually be passed as two bytes, similar to the Height and Name.

Don't underestimate how misunderstood this is. Even early versions of Power C (C Power) got it wrong! When a formal parameter was declared as *char*, the compiler got confused as to whether it was getting one byte or two, and generated inconsistent code [2]. This was fixed in the later releases.[3]

Many authorities on C programming style strongly discourage declaring formal parameters as type *char*, because it is inaccurate [3]. The compiler is supposed to know that it will actually be an *int*. Although it is changing, current standard versions of C provide no means for the intended type of a function parameter to be determined when the code for the function call is generated. Those *int* (even if they only involved a single *char*) expressions, therefore, would have to be assumed to be *ints* at the calling end. Discussion about this point raged for a good month on the Power C BBS about a year ago.

Function calls as parameters

Another apparent error may have been an intentional simplification on the part of the author. It is not correct to say that the value passed to a Power C function in the accumulator is al-

ways the number of bytes of arguments passed. Another value is also passed to the function. This is placed on the Power C runtime stack, the top of which is pointed to by ($1a,$1b). This value is the offset into the cassette buffer where the function's parameter list begins, and where the return value should be put. In most cases, this offset is zero, but if a function call is a parameter to another function, this offset will be non-zero. 'c$functinit' will pop this value off the C stack, and place it into the X register, where it can easily be used to access the correct area of the cassette buffer. The value passed in the accumulator is actually the upper bound of this area. (That is, the number of bytes in the parameter list, *plus* the offset of the start.) When the offset is zero, this value is obviously the same as the number of bytes in the parameter list. This is the general case, especially if the functions defined are "routines", rather than functions returning a useful value. When a function does return a value, this convention arranges that the return value of one function is already set up to be used as a parameter for the next.

Another observation has bothered me for some time. Perhaps it is unfair to single out David's routines, but they have brought it to mind. David's function "Slowkeys" is supposedly a general-purpose routine, but it makes a subtle assumption about its environment. It does an SEI, because it wants to inhibit interrupts, but then blithely does a CLI afterwards. Would not the following sequence be preferable: php; sei; [code]; plp? The plp at the end restores the previous state of the interrupt flag, rather than simply clearing it. This way if someone *happened* to call "Slowkeys" with interrupts already inhibited for some purpose, it wouldn't have an unexpected side effect for them! Not to worry too much. Even the C64 Kernal makes the assumption in several places that the calling routine doesn't have interrupts already inhibited.

Using an REU with Power C

A couple of quick tips for 1764 RAM disk users. The 1764 does not work with all of Power C in its standard distribution form, but it does speed a lot of things up, making the environment a lot more pleasant.

First 1764 tip: Don't you find it annoying when you are running a disk intensive program, using the RAM disk instead of a real disk drive? Your machine just sits there. Silently. Doing who knows what? Looping? Crashing? Locking up? "If only I could hear that disk!", you think. Well, just poke a volume value into the SID chip before you start the RAM disk activity and, lo and behold, you would swear at times you have a very quiet (but audible) hard disk at work for you! The sounds seem clearer after I have been playing with my music program (also written in Power C), but I haven't really analyzed the correlations. This works on my setup; I don't know if it will work in general. It is possible that it is partly interference with the monitor (lines output to the screen also seem to cause 'chirping' - though I *suspect* RAM gets swapped in for every CHKIN/CHKOUT). The address to poke is $d418 (54296) and 15 (maximum volume) is a good value to put there.

Another tip is a little less offbeat. I sometimes used to worry about which of my RAM disk files I had and had not saved to a real, live floppy! But then I noticed something. I never 'replace' a file on the RAM disk (using @), but always rename the old, then save the new, scratching the old one later when I feel safer. Well, it seems the 1764 RAMDOS doesn't create 'holes' in its directory when deleting files; thus the most recently changed files are always at the bottom of the directory listing. So, I took to creating a file named "----------" as the last file I transferred to my RAM disk in my startup procedure. Any files I modify end up below this 'bar', indicating they should be saved. From time to time I 'move' the bar to the bottom with a rename, copy, rename, delete sequence when I am sure I have properly archived everything so far.

Another question regarding the 1764 (and the RAMDOS provided with it). Is there anywhere to find a concrete list of known problems? Everyone hints at bugs, but it might be nice to have a verified list somewhere. My own worst observation, on RAMDOS 3.3, regards a simple-minded attempt to get around the lack of support for the concatenation option of the DOS copy command. I wrote an (admittedly inefficient) one-character-at-a-time CHKIN/CHRIN CHKOUT/CHROUT loop, and it substituted an incorrect character in the output every 256th character. When I changed it to a buffered loop (254 reads, 254 writes), the bug disappeared. Both programs worked correctly with a real disk drive. It smells like a subtle hardware/software timing bug, but it really needs confirming on someone else's hardware. There seem to be few 1764s in the stores around town, and owners seem less disposed to investigating such things than they were at one time, anyway.

I have written some examples to demonstrate the problem. The "concat.a" program works with a 1764, the "badcat.a" *should*, but does not. They need assembling with the C/ASSM assembler, and linking with the Power C linker to run in the Power C Shell environment. The principles they illustrate are quite simple, however, and it should be easy to convert them to use a different assembler, should anyone be so inclined.

"fred.*" are all programs that do nothing. I wrote "fred.c", and disassembled it in different ways (before and after linking) to illustrate how the Power C calling sequence works for nested function calls. "fred.a" is straight RAM disassembly. "fred.doc" has comments added to explain the code.

References

[1] *The C Programming Language*, Kernighan, B.W. and Ritchie, D.M., Prentice-Hall, 1978, p.183.

[2] The language definition specifies that this detail is implementation dependent. "Whether or not sign-extension occurs for characters is machine dependent, but it is guaranteed that a member of the standard character set is non-negative." Ibid. p.183.

[3] Ibid. pp. 39-40, for example.

Programming in GEOS

Entering the geoSphere...

by Francis G. Kostella

Francis G. Kostella is the author of the CIRCE strategy game, which runs under GEOS. He can be reached via CompuServe E-Mail (72220,3117) or on Q-Link as FGK.

GEOS is the first alternate operating system for the C64 that has gained any widespread acceptance. The C64 Kernal is familiar to assembly programmers but, unlike the C64 Kernal, the GEOS Kernal has not been widely documented and commented upon (see the references at the end of this article). This article will present enough information for the novice GEOS programmer to start programming in the GEOS environment, and a sample program that illustrates a few of GEOS's features.

The examples here were developed and tested using GEOS v1.2 and the Commodore MADS assembler with Bill Dixon's source editor and "assemfix" upgrade. The label names used below are are in upper case for clarity and are very similar to the standard BSW (Berkeley Softworks) labels. *[Mr. Kostella's source file contained labels of up to 12 characters. To simplify matters for users of other assemblers, the source has been converted to PAL format with six-character labels. The labels used were taken from Alex Boyce's Tech Manual and will, in all likelihood, be used in future GEOS programs published in* ***Transactor****. The original source file will be included on the* ***Transactor*** *disk for this issue. -Ed.]*

Getting started

The first hurdle is that all GEOS disk files have a different structure than that used by the Commodore DOS. This becomes obvious upon examining a directory entry on a GEOS disk.

```
          Sample Directory Entry
-----------------------------------------
$00 : -- -- c3 05 08 54 65 73 : .....tES
$06 : 74 20 46 69 6c 65 a0 a0 : T fILE
$0e : a0 a0 a0 a0 a0 05 00 00 :      ...
$16 : 06 57 0a 11 0f 1e 1a 00 : ........
-----------------------------------------
```

You'll notice that GEOS not only makes use of the formerly unused bytes, but also changes a few around to suit itself (see Table 1). Also note that because of these changes, REL files are not allowed under GEOS.

```
*************************************************************
* ------------------------------------------------------- *
* TABLE 1: FORMAT OF A GEOS DIRECTORY ENTRY               *
* ------------------------------------------------------- *
*                                                         *
* OFFSET INTO                                             *
*  DIR ENTRY   DESCRIPTION                                *
* ----------- ------------------------                    *
*     0       Commodore DOS file type                     *
*     1-2     If GEOS SEQ - points to track & sector      *
*                 of file's 1st block.                    *
*             If GEOS VLIR - points to track &            *
*                 sector of VLIR index table block.       *
*     3-18    16 char ASCII filename, padded.             *
*     19-20   Points to File Header's track & sector      *
*     21      GEOS File Structure. 0=SEQ  1=VLIR          *
*     22      GEOS File Type (see below).                 *
*     23-27   Last used: year/month/day/hour/minute.      *
*     28-29   Blocks in file                              *
*                                                         *
*    GEOS FILE TYPES                                      *
* ------------------------------------------------------- *
*    0  - not GEOS              7  - application data     *
*    1  - basic                 8  - font                 *
*    2  - assembly              9  - printer driver       *
*    3  - data                  10 - input driver         *
*    4  - system file           11 - disk drive           *
*    5  - desk accessory        12 - system boot          *
*    6  - application           13 - temporary            *
* ------------------------------------------------------- *
*************************************************************
```

The filename is stored in ASCII (thus the case of the characters appears inverted); PETSCII is not used in the GEOS system. Also, every GEOS file is of the C64 USR type, that is, the internal structure is user-determined.

The time and date are stamped into the 5 bytes before the block count (last 2 bytes). Bytes 19 and 20 point to the file's *Header Block*. Every GEOS program has a Header Block - a single sector, not directly connected to the file - that holds GEOS-specific information (the most important being the icon definition, the load address and the start address). An example Header Block is included in Program 2.

Probably the two most important of the extra bytes are bytes 21 and 22. These describe the GEOS file structure, and tell the Kernal what type of file it is. Byte 21, the File Structure Byte, is 0 if the file structure is sequential; that is, the file is stored sequentially on disk, as a PRG file would be. Unlike a PRG file, though, the load address, if any, is not stored as the first two bytes, but in the Header Block. The example program will be in sequential form.

When byte 21 is 1, the file structure is VLIR (Variable Length Indexed Record). Although the use of VLIR files is beyond the scope of this article, a few facts will help you explore their structure in more detail. When the file is of VLIR type, bytes 1 and 2 will not point to the file per se, but to a single-sector record index. The first two bytes of this index sector are always $00 and $ff, and the following 254 bytes are pairs of pointers to individual records. If a record's pointers are $00,$ff, then that record does not exist. A VLIR file may have up to 127 records (0-126). Each record is structured sequentially and may be any length. For example, a font file's index contains pointers to its various point sizes (0-48). So, if bytes 14 and 15 point to a valid track and sector, then they always point to the 6-point record.

Byte 22 of the directory entry is the GEOS file type (see Table 1), which should be familiar to the GEOS user. This byte tells the Kernal where and how to load a file. In this article we'll be writing a simple application that we can call from the Desk Top, and are thus concerned only with value 6, a *GEOS Application*. When the Kernal loads an application-type file, it will load it in place of the Desk Top and JSR to the start address given in the Header Block.

The reason for delving into the file structure is that most (if not all) assemblers do *not* output GEOS applications, but produce "binary" or object files. If we want to have our code run under the GEOS Kernal, we need a method of translating a standard object file to GEOS format. Thus, Program 1, "maketogeos".

Translating to GEOS

The process that "maketogeos" will go through to translate our file is as follows:

- find the file's directory entry
- make block 1 the Header Block, separating it from the program by saving the next track and sector pointers and changing them to $00,ff (end of file, $ff is last byte).
- change the file's load address into the icon dimensions (see the Header Block in Program 2).
- put the track and sector of this block into bytes 19 and 20 of the directory entry.
- put the previously saved track and sector pointers to block 2 into bytes 1 and 2 of the directory entry. This block is now the beginning of the file.
- Now write the new GEOS info to the directory entry, prompting for date and time.

As long as we structure our object file properly, GEOS will recognize it as a valid file when translated. Specifically, the Header Block has to be assembled at the beginning of the file, exactly 252 bytes before the beginning of our application code (remember, PRG files save the load address as the first two bytes, and they use 254 bytes per sector.)

Main loop

In its basic form, a GEOS application will usually consist of an initialization routine, a set of data tables, and a set of service routines. When our application is loaded, the Kernal will JSR to the start address held in the Header Block (bytes 75 and 76). This address will point to our initialization code, which will usually be called once to create menus, icons, graphics, and so on, all of which are defined by a set of data tables. The initialization code terminates with an RTS, which returns to the Kernal's MAIN LOOP. The MAIN LOOP just checks for user input and watches a set of IRQ process timers. If the user clicks on an icon, MAIN LOOP determines which icon was selected and calls the service routine associated with that icon. The service routine performs whatever action is required and then returns to MAIN LOOP.

The important thing to understand is that, in essence, we only have to write a set of subroutines, since all of our basic functions (IRQ, character printing, math, disk, graphics, etc.) are already there. Our code doesn't do anything until the user performs some action (or one of the processes times out).

At this point a few examples may make things clearer, but first a word about GEOS routines, variables, and constants.

The *GEOS Programmer's Reference Guide* (see the references) lists over 600 constants, 200 variables, and over 150 routines (called via a jump table at $C100-C2D5). Quite a bit to work with! Documenting even just the routines would fill many pages, but here we'll be concerned with just a few of them. The applicable constants' labels are not used in most of the included source code, but are explained in the comments. The variables are listed where they're introduced, with the exception of the zero-page registers. The Kernal routines make use of 16 two-byte pseudo-registers labeled R0 to R15, starting at bytes $02/03 (R0) and ending at bytes $20/21 (R15). Additionally, there are ten pseudo-registers not used by the Kernal, reserved for application use only. These are labeled A0 to A9. A0 is at $FB/FC, A1 is at $FD/FE, and the rest start at $70/71 (A2) and continue sequentially to $7E/7F (A9).

Menus

Most applications will want a menu, and this is a good place to start experimenting with GEOS' code structure.

Our initialization code will inform the Kernal that we are using a menu by placing the address of the menu definition table into pseudo-register R0 and calling the routine DOMENU. Let's look at an example:

```
    ldx <#OURMENU      ;lo
    ldy >#OURMENU      ;hi
    stx R0
    sty R0+1
    lda #1 ;leave pointer at this choice
    jsr DOMENU
```

The Kernal now expects to find a table at address OURMENU defining the menu structure. After drawing the menu, it will leave the mouse pointer on selection one (the second one). The first section of the menu table tells it where the menu is located on the hi-res screen, what type of menu it is and how many selections it displays. Our table might start like this:

```
OURMENU =*
    .byte $00          ;top
    .byte $0f          ;bottom
    .word $00          ;left
    .word $60          ;right
    .byte $02          ;type/items
```

The first four entries describe the outer borders of the menu, the origin of the hi-res screen being the upper left corner. The last byte is the number of menu items ORed with the menu type. The above example describes a horizontal menu with two items. There are three types of menus (it may be helpful to think of them by the bits they set):

$00 horizontal
$80 vertical
$40 constrains pointer to menu

Following this position table will be a selection table, one for each item. Immediately following the example above, our two selections might be:

```
    .word S1TEXT       ;addr of text
    .byte 128          ;sub-menu
    .word S1MENU       ;addr of submenu
    .word S2TEXT       ;addr of text
    .byte 0            ;menu action
    .word S2RTN        ;addr of rtn
```

The first entry of each table holds the address of a null-terminated ASCII text string that appears in the menu bar for that selection. The third table entry holds the address of the routine (or sub-menu table) that is called when that selection is chosen. The middle byte describes what to do when that item is selected:

$80 calls a submenu
$00 calls a service routine
$40 calls routine before displaying submenu; the routine exits with the submenu table address in R0.

Quite often, our main menu will call submenus. A submenu is set up with the same type of tables we have just shown, first the position/type/number then the individual entries. We can nest menus down four levels. Eventually, we'll want to call a service routine and/or roll up the menus displayed. We have three possibilities: REDOMENU, DOPREVIOUSMENU, and GOTOFIRSTMENU. Respectively, these re-enable the current menu, go back one level, and go to the first. Using our example service routine above:

```
S1RTN =*
    jsr REDOMENU  ;any of the three

    ... our service routine ...

    rts           ;back to main loop
```

When the menu is rolled up, the screen is recovered, so we usually want to use one of the three routines before changing the screen. Otherwise, if you were to print text where the menu was, it would be destroyed by the old screen.

A bit about graphics

GEOS uses two 8000-byte hi-res screens to display all text and graphics. The main screen is at $A000, and the secondary screen is at $6000. Our application code space is from $0400 to $5FFF, and we may optionally use the second screen for code. As mentioned above, GEOS has the ability to recover previously drawn graphics to its main hi-res screen. We'll not explain the process, but only mention that properly exiting the service routines for menus and dialog boxes will automatically recover anything that these structures may have overwritten.

We'll illustrate a few of the graphics routines shortly, but first we have to look at the formats used by GEOS to store graphic information for icons and bitmaps. Compacting graphic data saves code and disk space, not to mention disk access time. GEOS uses three different compaction formats; all three compact and uncompact scan lines, *not* the character cells typically used in C64 graphics. (Be aware that if you do any digging through GEOS data files, you'll find that VLIR geoPaint documents do store their data compacted into character cells, but that Photo Scraps and Photo Albums use scan lines. All compact their colour data immediately following the individual bitmaps.)

All three formats consist of a COUNT byte followed by one or more data bytes. These COUNT/data groups are repeated until the entire bitmap graphic is described.

Count	Description
000-127	Repeat next byte COUNT times.
128-220	First subtract 128; that gives the number of following bytes to use once each.
221-255	First subtract 219; that gives the number of bytes in the pattern following the 2nd byte. The second byte tells how many times

```
        the pattern is repeated. The pattern
        starts with the 3rd byte and is made
        up of the other two formats.
```

If that seems obscure, don't worry - we'll only use the first two formats in the examples here.

A few drawing commands

To draw a line between two points we call the routine DRAWLINE. Before calling the routine we need to put the coordinates of our endpoints into the pseudo-registers:

R3	x1 (0-319)
R11 (lobyte)	y1 (0-199)
R4	x2 (0-319)
R11+1 (hibyte)	y2 (0-199)

If the carry flag is set when calling DRAWLINE, the line is drawn in the foreground colour; if it's clear, the line is drawn in the background colour. Setting the sign flag recovers the bits from the secondary screen (and ignores the carry flag); clearing this flag draws on the main hi-res screen.

We draw a single point by calling the routine DRAWPOINT. The *x* value is put into R3, and the *y* value is put into the low byte of R11. The carry and sign flags operate the same as they do for DRAWLINE.

The RECTANGLE routine draws a solid rectangle using one of the Kernal fill patterns set by the routine SETPATTERN. FRAMERECTANGLE draws the outline of a rectangle using a pattern byte that describes the bits in the line ($FF, %11111111 would be a solid line, $55 is the pattern %01010101.) RECTANGLE and FRAMERECTANGLE expect the borders of the area to be described in these pseudo-registers:

R3	left (0-319)
R4	right (0-319)
R2 (lobyte)	top (0-199)
R2+1 (hibyte)	bottom (0-199)

The pattern byte for FRAMERECTANGLE is held in .A before the call is made. The following example draws a 100 by 100 bit rectangle in fill pattern 2, and puts a solid frame around it. We'll use the "inline-pass" form of RECTANGLE:

```
    lda #2              ;50% 'stipple'
    jsr SETPATTERN
    jsr I.RECTANGLE     ;inline call
    .byte 20            ;top
    .byte 120           ;bottom
    .word 45            ;left
    .word 145           ;right
; the borders for the frame are
; still held in R2-4, so...
    lda $ff                 ;solid
    jsr FRAMERECTANGLE
```

We'll mention just one more graphic command before moving on. BITMAPUP allows us to display a compacted bitmap on the hi-res screen. This routine also has an inline form, which we'll use in this example that puts a 40 by 40 bitmap in the upper left corner:

```
    jsr I.BITMAPUP   ;inline call
    .word YOURBITMAP ;address
    .byte 0  ;x pos in bytes
    .byte 0  ;y pos in pixels
    .byte 5  ;bitmap width in bytes
    .byte 40 ;bitmap height in pixels
```

You might be wondering what usefulness this call would have, if you don't have a compacted bitmap handy (at least not in .byte definitions for your assembler). A simple technique is to steal graphics from Photo Scraps. Photo Scraps are stored sequentially on disk and are already compacted. All we have to do is read in the data from the USR file and convert the bytes to hex (or any form our assembler can use). Or we might just tack a copy of the file on at the end of our code (being careful with labelling our bitmap's address). Remember that the colour data is compacted at the end of the bitmap.

Icons

In some ways, icons are easier to program than are menus. Once again, we need to put the address into R0, and call our set-up routine. This will be part of our initialization code:

```
    ldx <#OURICONS      ;lo
    ldy >#OURICONS      ;hi
    stx R0
    sty R0+1
    jsr DOICONS
```

Again, the Kernal expects to find a table defining the icons at address OURICONS. It is importantly to remember that *every* application must have at least one icon; it may be invisible and it may do nothing, but it must be defined or strange things will happen. The example code shows how to define a 'dummy' icon.

The first part of our icon table is very simple:

```
OURICONS =*
    .byte 2         ;number of icons
    .word 10        ;x pos. mouse
    .byte 10        ;y pos. mouse
```

This tells the Kernal that we're defining two icons, and to leave the mouse pointer at position 10,10 on the hi-res secreen. Now it's time for the individual icon entries. Following the example above:

```
    .word ICON0GRAFIC ;addr of bitmap
    .byte 35          ;horizontal byte
    .byte 160         ;vertical pixel
```

```
        .byte 2             ;bytes wide
        .byte 8             ;pixels high
        .word ICON0RTN      ;addr of srvc rtn

        .word ICON1GRAFIC ;bitmap addr
        .byte 5           ;horizontal byte
        .byte 20          ;vertical pixel
        .byte 4           ;bytes wide
        .byte 16          ;pixels high
        .word ICON1RTN ;addr of service rtn
```

The first entry in each icon's table holds the address of the icon's graphic data, stored in the compaction formats outlined above (see the source code and the section on dialog boxes for a simple example). The second entry holds a value from 0 to 39, and indicates, in bytes, the distance from the left of the screen to the starting position of the icon's picture. (Think of them as character cells; each byte equals 8 pixels. The left edge of an icon, as far as I've been able to determine, *always* begins on a cell boundary.) The third entry is the number (0-199) of pixels (or scan lines) down to draw the graphic. Using the example above, icon 0 would appear in the lower right area of the screen, and icon 1 would appear in the upper left area.

The fourth entry is the width of the icon graphic in bytes, the fifth entry is the icon's pixel height. In the example above, icon 0 is 16 pixels wide by 8 pixels high, icon 1 is 32 by 16.

The final entry in each icon's table holds the address of the icon's service routine. These routines can do almost anything, even define new icons. Often they will finish with an RTS to MAIN LOOP. When a user clicks on an icon, the Kernal returns the number of the selected icon (0-30) in the low byte of pseudo-register R0. Thus we could have a number of icons share the same routine that, when called, checks R0 first then chooses an appropriate action.

Dialog boxes

A dialog box (DB) is a small window put on the screen to prompt the user for input or warn about possibly unexpected conditions. A familiar example from Desk Top is the DB used to rename a file. Calling a DB causes the Kernal to save most of the state of the application. We can run the DB, as if it were itself a small application, without affecting the rest of the program (unless we need to).

Once again, a table is used, this time to define the structure of a DB. We run the DB by passing the address of the table in R0 and calling DODLGBOX. When the DB is finished, R0 returns the number of the icon (if a system icon), or user-supplied value that terminated the DB. A dialog box table is made up of a number of DB commands, and is terminated by a zero byte.

The very first entry in the DB table is the position byte. The lower bits specify the number of the Kernal fill pattern that makes up the shadow box. If the high bit of the position byte is 1, the DB's dimensions are the default dimensions (as are most of the Desk Top DBs), and the very next byte is the beginning of the next DB command. If the high bit is 0, the the next four entries are the DB's dimensions. See the source code for an example.

After the position, we may define up to eight icons using the predefined DB system icons or user-defined icons. We may also use as many non-icon DB commands as we wish. Six DB system icons are already defined by the Kernal. We only have to enter their positions; the Kernal will take care of the rest and, upon exiting the DB, will return the icon's number in R0 if it is selected. Here are the six system icons:

1	OK	4	NO
2	Cancel	5	Open
3	Yes	6	Disk

These should be familiar to all GEOS users. All six of them are 6 bytes wide and 16 lines deep. Immediately following any of the six in a DB table would be two position offset bytes. The first one is the number of bytes to position the icon from the left of the DB, the second is the offset from the top in scan lines. Here is a simple, complete DB table using the OK icon:

```
OURDBTABLE =*
   .byte $01      ;default pos./solid shadow
   .byte $01      ;OK icon command
   .byte $02      ;16 pixel x offset
   .byte $10      ;16 scanlines y offset
   .byte 0        ;terminate table
```

This will simply put up a DB with an *OK* icon and do nothing, until the user clicks on OK. In this instance, when OK is selected, the Kernal returns to the caller with $01 (OK) in R0. If we had put up an *Open* icon instead, R0 would hold $05 upon return.

There are also a number of DB commands used to print text strings or to define your own icons, among other things. Most of them, however, require familiarity with routines and Kernal methods not presented in this article. We will examine only two here.

To print a text string in a DB, we use the DB command $0B (11) in the DB table. It is followed by two position offset bytes, as used above. The final entry is the two byte address of a null-terminated string. To define our own icons, we use the command byte $12 (18). It, too, is followed by two position offset bytes, and a two byte address, this time pointing to an icon table. This icon table is the same as a regular icon table except that the position has already been set by the DB table, so the two bytes normally used for this purpose are made null. Here is a complete example of these new commands:

```
OURDBTABLE =*
   .byte $01      ;default/solid
   ;
   .byte $0b      ;DB text string command
```

```
        .byte $01,$0d    ;x bytes, y lines
        .word OURTEXT    ;string address
        ;
        .byte $12        ;non-standard icon
        .byte $03,$16    ;x bytes, y lines
        .word OURICON    ;icon table address
        .byte 0          ;end of table
        ;
OURTEXT =*
        .byte 'A SIMPLE STRING'
        .byte 0
        ;
OURICON =*       ;similar to regular icon
        .word OURICONPIC ;graphic address
        .byte 0          ;x set above
        .byte 0          ;y set above
        .byte $01        ;width in bytes
        .byte $08        ;height in lines
        .word OURSVCRTN  ;service address
;
OURICONPIC =*
        .byte $88    ;format 2/8 bytes follow
        .byte %11111111 ;a very simple icon
        .byte %10000001
        .byte %10000001
        .byte %10000001
        .byte %10000001
        .byte %10000001
        .byte %10000001
        .byte %10000001
        .byte %11111111
;
OURSVCRTN =*                ;service routine
        lda #$10            ;value to be
        sta SYSDBDATA       ;placed in R0
        jmp RSTRFRMDIALOG ;exit to caller
```

A few things about OURSVCRTN need to be explained. As we've said, exiting from a DB via one of the sytem DB icons will leave that icon's number in R0. But the Kernal knows nothing about our icons, and doesn't exit the DB when they are called. The Kernal *does*, however, provide a method of exiting a DB and passing information back to the caller.

We place the value we want into the variable SYSDBDATA, and JMP to the routine RSTRFRMDIALOG. This allows the Kernal to return the state of the application back to where it was before we entered the DB, then place our value into R0. If we were to, say, draw a graphic on the screen from our service routine, when the Kernal recovers the screen under the DB, our graphic might be erased. But if we pass a value to R0 (via SYSDBDATA), we can recover the screen, then draw our graphic.

Text in GEOS

There are a number of complexities dealing with printing text to the hi-res screen. Here I'll just present the two main character printing routines, and a brief description of potential problems.

The PUTSTRING routine will print a null-terminated string to the screen; it is probably the most widely used of the GEOS text routines. We first place the horizontal position (0-319) into R11, and the vertical position into the low byte of R1. We stuff the test string's address into R0 and JSR PUTSTRING. Alternately, we can use the in-line form:

```
        jsr I.PUTSTRING
        .word 20              ;x position
        .byte 20              ;y position
        .byte "A SIMPLE STRING"
        .byte 0               ;null terminated
```

The other often-used routine, PUTDECIMAL, is used to print 16-bit numbers to the screen. The set-up is similar to PUTSTRING (*x* and *y* go into R11 and R1), but here we put the number to be printed into R0, and load the accumulator with a format byte. The format byte determines how the number will be printed. If bit seven is 1, the number is printed left justified. If bit seven is 0, the number is printed right justified. If bit six is 1, leading zeroes are suppressed. If bit six is 0, leading zeroes are printed. If we are using right jusification, the lower bits hold the pixel width of the field the number is printed in. An example of PUTDECIMAL is included in the source code accompanying this article.

Be aware of a potential problem that may crop up when using PUTSTRING. Any text to be printed that goes beyond the screen borders won't be printed. There is a vector the Kernal calls when attempting to print beyond the borders; its name is STRINGFAULTVECTOR. The Kernal will only JSR to this address if it is non-zero. The routine pointed to by this vector might perform a word wrap and move to the next line, or scroll up the screen, depending on which border was crossed. An entire "print at" routine is a bit beyond our scope here, but would be a very useful module for the GEOS programmer. Perhaps such a module will appear in a future *Transactor.*

Finishing up

To exit our application we use the call JMP ENTERDESKTOP. This re-initializes the system and returns us to DeskTop. That's it! A complete GEOS application.

References

Two books you'll find invaluable for writing GEOS programs:

Berkeley Softworks' *The Official GEOS Programmer's Reference Guide*, Bantam Books, 1987 ($20 US/$25 Cdn.)

Alexander Boyce's *GEOS Programmers Reference Guide*, Alexander Boyce, 1986.

Alex Boyce wrote his shareware guide by dissassembling the entire GEOS Kernal, and it covers just about everything in its 95 pages. Omissions are few, and I've yet to find a single error. The only problem is that all the label names are six charac-

ter non-standard names, and even this is only a problem when using both this and the BSW guide in tandem. If you get a copy of this guide, send Mr. Boyce a donation - efforts like this need to be supported. *[Alex Boyce's manual is available from Mystic Jim (see NewsBRK). -Ed.]*

The BSW guide was written by the developers of GEOS, and in my opinion should have been better. Though all the calls are presented, and most descriptions are understandable, the downfall of this guide is the numerous typographical errors, the items mentioned but left out, and the few examples, none of which will work in the form presented. On the other hand, if you verify the unclear sections with Alex Boyce's manual, you should have very few problems. BSW is in the process of rewriting this guide, and the second edition should be in much better shape. I have no idea when it's due out; if they give it the attention it deserves, it may be a while.

Program 1: "maketogeos"

```
CN  100 rem  save"maketogeos",8
HG  110 rem originally part of larger prg
JD  120 dims%(255)
BE  130 gosub370
MI  140 end
KA  150 :
EJ  160 rem disk error
BN  170 input#15,en,em$,et,es:ifen=0thenreturn
NI  180 print"{rvs} disk error {rvs off}"en,em$,et,es
AJ  190 gosub250:return
MD  200 :
HA  210 open 15,8,15,"i0":rem <<open all>>
FJ  220 gosub170
OA  230 open 2,8,2,"#"
MA  240 return
JG  250 close2 :rem    << close all >>
IM  260 print#15,"i0"
EP  270 forx=0to2000:next
FB  280 close15:return
GJ  290 :
EI  300 rem << read sector >> t,s,s%(255)
MF  310 print"reading trk:";t;"sec:";s
FA  320 print#15,"u1";2;0;t;s
MO  330 gosub170:fori=0to255:get#2,b$
GH  340 s%(i)=asc(b$+chr$(0)):next:return
CN  350 :
AL  360 rem convert a c64 file to geos
KF  370 print"input filename":print:inputf$:iff$=""thenend
CH  380 forx=0to15:f$=f$+chr$(160):next:f$=left$(f$,16)
DG  390 gosub210:gosub 600:rem dir
KI  400 t=d1:s=d2:gosub310:rem get info
FG  410 e4=s%(0):e5=s%(1):rem link
GM  420 s%(0)=0:s%(1)=255:rem /change
MM  430 s%(2)=3:s%(3)=21 :rem /1st 4
HB  440 gosub690:rem write block
KJ  450 t=e1:s=e2:gosub310:rem get dir
CL  460 gosub 790:rem dir entry info
MN  470 s%(e3)=131:rem user/c=64
FD  480 s%(e3+1)=e4:s%(e3+2)=e5:rem vlir
MD  490 s%(e3+19)=d1:s%(e3+20)=d2:reminfo
FG  500 s%(e3+21)=0:rem seq/geos
KE  510 s%(e3+22)=6:rem application/geos
KE  520 s%(e3+23)=t1
GF  530 s%(e3+24)=t2
CG  540 s%(e3+25)=t3
OG  550 s%(e3+26)=t4
KH  560 s%(e3+27)=t5
IG  570 gosub690:gosub250:return
IL  580 :
CA  590 rem find a dir entry
AP  600 t=18:s=1:gosub310
OM  610 fori=5to229step32
PK  620 g$="":forj=0to15
AA  630 g$=g$+chr$(s%(i+j)) :next
MA  640 ifg$=f$thend1=s%(i-2):d2=s%(i-1):e1=t:e2=s:e3=i-3:return:
          rem e3=filetype
AE  650 next:ifs%(0)<>0thent=s%(0):s=s%(1):gosub310:goto610
DC  660 print"{rvs}  not found  {rvs off}":return
CB  670 :
JO  680 rem write sector to disk
JD  690 print"writing trk:";t;"sec:";s
FO  700 print#15,"b-p";2;0
CI  710 fori=0to255
OP  720 print#2,chr$(s%(i));
ON  730 next
NK  740 print#15,"u2";2;0;t;s
FM  750 gosub170:return
MG  760 :
GH  770 :
DH  780 rem get dir entry info
HF  790 print"{down}{down}dir. entry information"
DJ  800 input"year :";t1:ift1>99then800
EE  810 input"month:";t2:ift2>12then810
NM  820 input"day  :";t3:ift3>31then820
NJ  830 input"hour :";t4:ift4>23then830
NF  840 input"min. :";t5:ift5>59then840
CC  850 print"file:";f$:print"date:"t1;"/";t2;"/";t3;
          "  time:";t4;":";t5:poke198,0
FK  860 print"do you wish to change info (y/{rvs}n{rvs off}) ?":
          inputk$:ifk$="y"then790
CI  870 return
```

Program 2: "geosdemo.pal"

```
HK  100 open 2,8,2,"0:geosdemo,p,w"
PD  110 sys 700
JI  120 .opt o2
IP  130 ;
IG  140 ;f.g.kostella 12/10/87
MA  150 ;
EJ  160 *= $0304
AC  170 ;
FP  180 ;zpage pseudo-registers
ED  190 ;
ON  200 r0 = $02
II  210 r01 = $02
GI  220 r0h = $03
FA  230 r1 = $04
BL  240 r11 = $04
PK  250 r1h = $05
MG  260 r11 = $18
BC  270 r111 = $18
IC  280 r11h = $19
IJ  290 ;
NK  300 ;geos routines
MK  310 ;
BA  320 menu = $c151 ;domenu
NH  330 drwmnu = $c193 ;redomenu
LO  340 clsmnu = $c190 ;dopreviousmenu
FJ  350 cmenus = $c1bd ;gotofirstmenu
KC  360 line = $c130 ;drawline
FK  370 setpat = $c139 ;setpattern
MJ  380 plot = $c133 ;drawpoint
BL  390 pfill = $c124 ;rectangle
LC  400 pfill2 = $c19f ;i.rectangle
BI  410 pbox = $c127 ;framerectangle
JG  420 pbox2 = $c1a2 ;i.framerectangle
LK  430 cbox = $c124 ;bitmapup
HD  440 cbox2 = $c1ab ;i.bitmapup
LI  450 cboxes = $c15a ;doicons
```

```
ML  460 window = $c256 ;dodlgbox
PO  470 clswin = $c2bf ;rstrfrmdialog
AG  480 dsptxt = $c148 ;putstring
AD  490 dsptx2 = $c1ae ;i.putstring
EP  500 dspnum = $c184 ;putdecimal
BN  510 restrt = $c22c ;enterdesktop
OH  520 ;
PF  530 sfvec = $84ab ;stringfaultvector
FN  540 sysdb = $851d ;sysdbdata
MJ  550 ;
FB  560 ;----------------------------
HM  570 ;header block starts at $0304
GI  580 ;ram-based assemblers may need
GP  590 ;to change start address.
ND  600 ;----------------------------
AN  610 ;-assemble the header block here-
OB  620 ;             -note-
KB  630 ;1st 4 bytes commented out here
KD  640 ;they will be placed in the
JH  650 ;geos file header by "maketogeos"
KF  660 ;.byte 0,255 ; 1 sector
AB  670 ;.byte 3,21  ; 3x21 icon
OI  680 ;----------------------------
FE  690 ;define icon to appear on desk top
DD  700 .byte $bf ;$80 (straight bitmap) + 63 data bytes
LK  710 .byte %11111111,%11111111,%11111000
HI  720 .byte %10000000,%00000000,%00001000
BJ  730 .byte %10000000,%00000000,%00001000
NL  740 .byte %10011101,%11011101,%11001111
HL  750 .byte %10001001,%00010000,%10001111
HM  760 .byte %10001001,%11001000,%10001111
NM  770 .byte %10001001,%00000100,%10001111
PN  780 .byte %10001001,%11011100,%10001111
DN  790 .byte %10000000,%00000000,%00001111
NN  800 .byte %10000000,%00000000,%00001111
OP  810 .byte %10011101,%11010001,%11001111
IP  820 .byte %10010000,%10010001,%00001111
MA  830 .byte %10011100,%10010001,%11001111
MA  840 .byte %10010000,%10010001,%00001111
GC  850 .byte %10010001,%11011101,%11001111
JB  860 .byte %10000000,%00000000,%00001111
DC  870 .byte %10000000,%00000000,%00001111
LF  880 .byte %11111111,%11111111,%11111111
OF  890 .byte %00011111,%11111111,%11111111
IG  900 .byte %00011111,%11111111,%11111111
CH  910 .byte %00011111,%11111111,%11111111
OA  920 ;
LP  930 .byte $83 ;c64 filetype usr
MO  940 .byte 6 ;application
ID  950 .byte 0 ;geos seq file
GD  960 ;
HJ  970 .word saddr ;load start addr
LK  980 .word eaddr ;load end addr
EP  990 .word start ;start addr jump
OF 1000 ;
JE 1010 .asc "filename    v1.1" ;perm name string
OA 1020 .byte 0,0,0,0 ;
BG 1030 .asc "author name     "
GI 1040 ;
AI 1050 ;the rest of the header block
LM 1060 ;is not used in this file
EK 1070 ;
NB 1080 ; ----------------------------
MK 1090 ;ram based assemblers change addr
CD 1100 *= $0400
LD 1110 ; ----------------------------
GN 1120 ;
OB 1130 saddr =* ;save start
KN 1140 start =*
EP 1150 ;
MA 1160 ;   clean screen
NH 1170 lda #0
DH 1180 jsr setpat
CB 1190 jsr pfill2
EK 1200 .byte 0
OM 1210 .byte 199
JI 1220 .word 0
AB 1230 .word 319
FJ 1240 lda #$ff
IN 1250 jsr pbox
CG 1260 ;
MJ 1270 ;1 icon required at all times, so...
GH 1280 ;
MH 1290 ldx #<dummy ;dummy until
CN 1300 ldy #>dummy ;we need one
GL 1310 stx r0l
IL 1320 sty r0h
GA 1330 jsr cboxes
OJ 1340 ;  menus
OP 1350 ldx #<ourmnu
IA 1360 ldy #>ourmnu
CP 1370 stx r0l
EP 1380 sty r0h
LF 1390 lda #1
CG 1400 jsr menu
LJ 1410 ; that's all!, rts to main loop
IH 1420 rts
GJ 1430 ;============================
AC 1440 dummy =*
CK 1450 .byte 1 ;# of icons
KN 1460 .word 319 ;leave mouse x pos,
HD 1470 .byte 199 ;y pos
OD 1480 ;
JI 1490 .word 0 ;icon bitmap addr
OK 1500 .byte 36,1 ;h pos.byte(/8),v pos. pixel
JC 1510 .byte 1,1 ;w+h
DE 1520 .word 0 ;dispatch rtn
MK 1530 ;============================
BE 1540 ;... menu structure...
PE 1550 ourmnu =*
OB 1560 .byte 0 ;main top
OO 1570 .byte 13 ;main bottom
AJ 1580 .word 0 ;main left
GJ 1590 .word 80 ;main right
EH 1600 .byte 2 ;horz ($00) or'ed w/ # menu items
AM 1610 ;
AF 1620 .word filtxt
GC 1630 .byte $80 ;sub menu constant
MB 1640 .word filmnu ;rtn
IO 1650 ;
PL 1660 .word optxt
GG 1670 .byte $80
PK 1680 .word opmnu
AB 1690 ;
HI 1700 ;text for main selections
JD 1710 filtxt .asc "file"
MK 1720 .byte 0
GG 1730 optxt .asc "operations"
AM 1740 .byte 0
ME 1750 ;
PM 1760 ;..submenus...
AG 1770 ;
HO 1780 filmnu =*
BM 1790 .byte 13
ON 1800 .byte 27
HN 1810 .word 0
AI 1820 .word 33
NH 1830 .byte $81 ;vert ored w/ # items
GK 1840 ;
MB 1850 .word filxit
BL 1860 .byte 0 ;menu action
CP 1870 .word doexit ;rtn
OM 1880 ;
PD 1890 filxit .asc "quit"
AG 1900 .byte 0
MO 1910 ;
```

```
DG  1920 doexit =*
EG  1930 jmp restrt
KA  1940 ;
OB  1950 opmnu =*
DF  1960 .byte 13,55 ;top,bot
HO  1970 .word 23,80 ;left,right
GB  1980 .byte $83 ;vertical or'd w/ #
MD  1990 ;
DI  2000 .word op0txt
HE  2010 .byte 0 ;menu action
LH  2020 .word op0rtn
EG  2030 ;
PK  2040 .word op1txt
GP  2050 .byte 0
JD  2060 .word mover
MI  2070 ;
LN  2080 .word op2txt
OB  2090 .byte 0
LG  2100 .word sizer
EL  2110 ;
BP  2120 op0txt .asc "pattern"
GE  2130 .byte 0
BP  2140 op1txt .asc "mover"
KF  2150 .byte 0
HA  2160 op2txt .asc "sizer"
OG  2170 .byte 0
PA  2180 ;-----------------------
MD  2190 ourpat .word 0
OA  2200 ;
IE  2210 op0rtn =*
AG  2220 jsr cmenus
MC  2230 ;
AE  2240 lda ourpat
DP  2250 and #%00011111
CJ  2260 sta ourpat
FL  2270 jsr setpat
EF  2280 jsr pfill2
FL  2290 .byte 13
AB  2300 .byte 199
LM  2310 .word 0
CF  2320 .word 319
HN  2330 lda #$ff
KB  2340 jsr pbox
EK  2350 ;
DP  2360 jsr dsptx2
PK  2370 .word 92
DA  2380 .byte 10
HI  2390 .asc "pattern:    "
EF  2400 .byte 0
AO  2410 ;
OB  2420 ldx #132
JM  2430 ldy #0
BA  2440 stx r111
PP  2450 sty r11h
JH  2460 ldy #10
CC  2470 sty r1+1
MI  2480 ldx ourpat
FA  2490 ldy #0
IH  2500 stx r0
IE  2510 sty r0+1
EH  2520 lda #%11000000
IL  2530 jsr dspnum
CG  2540 ;
GJ  2550 inc ourpat
MO  2560 rts
AI  2570 ;
GP  2580 ;-------------------------
NB  2590 ;values used to add to pos bytes
KE  2600 dbtop .byte 0
BK  2610 dbbot .byte 0
DH  2620 dbleft .byte 0
LJ  2630 dbrght .byte 0
AN  2640 ;=====
IO  2650 mover =*
IB  2660 jsr cmenus
EO  2670 ;
HK  2680 dodb =*
JI  2690 jsr clradr
JE  2700 ldx #<dbtab
DF  2710 ldy #>dbtab
ID  2720 stx r01
KD  2730 sty r0h
JG  2740 jsr window
AI  2750 lda r0 ;returned by db
CJ  2760 bmi ours
EF  2770 ;  its 'ok'
IM  2780 rts
MF  2790 ;
AJ  2800 ours =*
CI  2810 cmp #$82
JA  2820 bcs ours1
NP  2830 lda #2
CL  2840 sta dbtop
GO  2850 sta dbbot
IF  2860 jsr dbsub
AG  2870 jmp ours4
GL  2880 ;
KK  2890 ours1 =*
AO  2900 cmp #$83
EG  2910 bcs ours2
HF  2920 lda #2
GM  2930 sta dbleft
MO  2940 sta dbrght
CL  2950 jsr dbsub
KL  2960 jmp ours4
AB  2970 ;
GA  2980 ours2 =*
OD  2990 cmp #$84
PL  3000 bcs ours3
BL  3010 lda #2
GG  3020 sta dbtop
KJ  3030 sta dbbot
EJ  3040 jsr dbadd
EB  3050 jmp ours4
KG  3060 ;
CG  3070 ours3 =*
HP  3080 lda #2
GG  3090 sta dbleft
MI  3100 sta dbrght
KN  3110 jsr dbadd
GK  3120 ;
AK  3130 ours4 =*
HA  3140 jsr dspval
PL  3150 jmp dodb
OM  3160 ;
LJ  3170 ;---------------------------
IC  3180 ;use the same db, process
FJ  3190 ;the results differently
GP  3200 ;
CC  3210 sizer =*
IE  3220 jsr cmenus
EB  3230 ;
JO  3240 dodbz =*
JL  3250 jsr clradr
JH  3260 ldx #<dbtab
DI  3270 ldy #>dbtab
IG  3280 stx r01
KG  3290 sty r0h
JJ  3300 jsr window
AL  3310 lda r0 ;returned by db
MB  3320 bmi oursz
OO  3330 rts
CI  3340 ;
IM  3350 oursz =*
IK  3360 cmp #$82
DO  3370 bcs ours1z
```

```
DC  3380 lda #2
IN  3390 sta dbtop
EH  3400 jsr dbsub
JF  3410 jsr clradr
LE  3420 lda #2
KC  3430 sta dbbot
EC  3440 jsr dbadd
IF  3450 jmp ours4z
KP  3460 ;
KK  3470 ours1z =*
EC  3480 cmp #$83
MF  3490 bcs ours2z
LJ  3500 lda #2
KA  3510 sta dbleft
MO  3520 jsr dbsub
BN  3530 jsr clradr
DM  3540 lda #2
OE  3550 sta dbrght
MJ  3560 jsr dbadd
AN  3570 jmp ours4z
CH  3580 ;
EC  3590 ours2z =*
AK  3600 cmp #$84
FN  3610 bcs ours3z
DB  3620 lda #2
IM  3630 sta dbtop
MO  3640 jsr dbadd
JE  3650 jsr clradr
LD  3660 lda #2
KB  3670 sta dbbot
MI  3680 jsr dbsub
IE  3690 jmp ours4z
KO  3700 ;
OJ  3710 ours3z =*
HH  3720 lda #2
GO  3730 sta dbleft
AF  3740 jsr dbadd
NK  3750 jsr clradr
PJ  3760 lda #2
KC  3770 sta dbrght
AP  3780 jsr dbsub
EE  3790 ;
KP  3800 ours4z =*
FK  3810 jsr dspval
HL  3820 jmp dodbz
MG  3830 ;
DD  3840 ;----------db subs--------------
AI  3850 ;
JN  3860 clradr =*
JA  3870 lda #0
CM  3880 sta dbtop
GP  3890 sta dbbot
AJ  3900 sta dbleft
GL  3910 sta dbrght
MD  3920 rts
AN  3930 ;
NJ  3940 dbsub =*
PO  3950 sec
OM  3960 lda dbtab+1 ;top of db
OO  3970 sbc dbtop
DC  3980 sta dbtab+1
HB  3990 sec
EJ  4000 lda dbtab+2 ;bot of db
AE  4010 sbc dbbot
OE  4020 sta dbtab+2
PD  4030 sec
KL  4040 lda dbtab+3 ;left of db
IP  4050 sbc dbleft
JH  4060 sta dbtab+3
IE  4070 lda dbtab+4
LO  4080 sbc #0
KJ  4090 sta dbtab+4
FI  4100 sec
OE  4110 lda dbtab+5 ;right of db
KF  4120 sbc dbrght
FM  4130 sta dbtab+5
EJ  4140 lda dbtab+6
BD  4150 sbc #0
GO  4160 sta dbtab+6
GD  4170 rts
KM  4180 ;
CE  4190 dbadd =*
ON  4200 clc
IM  4210 lda dbtab+1 ;top of db
KM  4220 adc dbtop
NB  4230 sta dbtab+1
GA  4240 clc
OI  4250 lda dbtab+2 ;bot of db
MB  4260 adc dbbot
IE  4270 sta dbtab+2
OC  4280 clc
EL  4290 lda dbtab+3 ;left of db
EN  4300 adc dbleft
DH  4310 sta dbtab+3
CE  4320 lda dbtab+4
HM  4330 adc #0
EJ  4340 sta dbtab+4
EH  4350 clc
IE  4360 lda dbtab+5 ;right of db
GD  4370 adc dbrght
PL  4380 sta dbtab+5
OI  4390 lda dbtab+6
NA  4400 adc #0
AO  4410 sta dbtab+6
AD  4420 rts
EM  4430 ;
OM  4440 ;
AF  4450 dspval =*
CO  4460 ;
BG  4470 lda #0
HF  4480 jsr setpat
GP  4490 jsr pfill2
LI  4500 .byte 1
JF  4510 .byte 11
OO  4520 .word 239
AP  4530 .word 318
CD  4540 ;
EG  4550 ldx #210
LB  4560 ldy #0
DF  4570 stx r111
BF  4580 sty r11h
LM  4590 ldy #10
KI  4600 sty r1h
HL  4610 ldx dbtab+1
HF  4620 ldy #0
OK  4630 stx r01
AL  4640 sty r0h
GM  4650 lda #%11000000
KA  4660 jsr dspnum
EL  4670 ;
AA  4680 ldx #235
NJ  4690 ldy #0
FN  4700 stx r111
DN  4710 sty r11h
IC  4720 ldx dbtab+2
FM  4730 ldy #0
ID  4740 stx r0
OB  4750 sty r0h
ED  4760 lda #%11000000
IH  4770 jsr dspnum
CC  4780 ;
FA  4790 ldx #4
NA  4800 ldy #1
DE  4810 stx r111
BE  4820 sty r11h
JJ  4830 ldx dbtab+3
```

```
KK  4840 ldy dbtab+4
GK  4850 stx r0
MI  4860 sty r0h
CK  4870 lda #%11000000
GO  4880 jsr dspnum
AJ  4890 ;
KB  4900 ldx #29
LH  4910 ldy #1
BL  4920 stx r11l
PK  4930 sty r11h
NA  4940 ldx dbtab+5
OB  4950 ldy dbtab+6
EB  4960 stx r0
KP  4970 sty r0h
AB  4980 lda #%11000000
EF  4990 jsr dspnum
EH  5000 rts
HH  5010 ;-----------------------------
FM  5020 dbtab =*
MB  5030 ;
PE  5040 .byte $01 ;pos/shadow patrn
AD  5050 ;
KP  5060 .byte 50 ;top
CA  5070 .byte 86 ;bott
BD  5080 .word 48 ;left
FM  5090 .word 120 ;right
CG  5100 ;
OL  5110 .byte 1 ;ok
KB  5120 .byte 1 ;x byt
AK  5130 .byte 16 ;y pixel
KI  5140 ;
GA  5150 .byte $12 ;user icon
EM  5160 .byte 1 ;x offset
IN  5170 .byte 4 ;y offset
AE  5180 .word db1 ;addr of icon table
ML  5190 ;
GB  5200 .byte $12
LD  5210 .byte 3,4
PN  5220 .word db2
EO  5230 ;
OD  5240 .byte $12
JG  5250 .byte 5,4
LA  5260 .word db3
MA  5270 ;
GG  5280 .byte $12
HJ  5290 .byte 7,4
HD  5300 .word db4
ED  5310 ;
IC  5320 .byte 0 ;end
JA  5330 ;-------------------------
FF  5340 ;db user icon tables, graphics
MO  5350 ;& service routines for mover
GG  5360 ;
GB  5370 db1 =*
AI  5380 .word db1bit ;addr of picture data
OI  5390 .byte 0,0 ;x,y-already set!
AD  5400 .byte 1 ;bytes wide
GA  5410 .byte 8 ;pixels hi
LE  5420 .word dodb1 ;addr of svc rtn
MK  5430 ;
GJ  5440 db1bit =*
AM  5450 ;
BE  5460 .byte $88 ;format 2, use the next 8 bytes
FA  5470 .byte %11111111
IA  5480 .byte %11100111
PA  5490 .byte %11000011
GB  5500 .byte %10000001
GC  5510 .byte %11100111
AD  5520 .byte %11100111
KD  5530 .byte %11100111
LE  5540 .byte %11111111
EC  5550 ;
HK  5560 dodb1 =*
ID  5570 ;
LA  5580 lda #$81
IG  5590 sta sysdb
MN  5600 ;  and get out
KL  5610 jmp clswin
PH  5620 ; ------------------------
OB  5630 db2 =*
DF  5640 .word db2bit
JL  5650 .byte 0,0,1,8
MK  5660 .word dodb2
MJ  5670 ;
KI  5680 db2bit =*
CC  5690 .byte $88
LO  5700 .byte %11111111
CP  5710 .byte %11101111
KP  5720 .byte %11001111
MP  5730 .byte %10000001
GA  5740 .byte %10000001
IB  5750 .byte %11001111
EC  5760 .byte %11101111
BD  5770 .byte %11111111
KA  5780 ;
CM  5790 dodb2 lda #$82
KD  5800 sta sysdb
CI  5810 jmp clswin
CD  5820 ;
MD  5830 ;
EP  5840 db3 =*
JC  5850 .word db3bit
LI  5860 .byte 0,0,1,8
AI  5870 .word dodb3
OG  5880 ;
AG  5890 db3bit =*
EP  5900 .byte $88
NL  5910 .byte %11111111
AM  5920 .byte %11100111
KM  5930 .byte %11100111
EN  5940 .byte %11100111
IN  5950 .byte %10000001
FO  5960 .byte %11000011
CP  5970 .byte %11100111
DA  5980 .byte %11111111
MN  5990 ;
DG  6000 dodb3 =*
AP  6010 ;
LM  6020 lda #$83
AC  6030 sta sysdb
IG  6040 jmp clswin
NC  6050 ; ------------------------
CC  6060 ;
MC  6070 ;
IO  6080 db4 =*
NB  6090 .word db4bit
LH  6100 .byte 0,0,1,8
CH  6110 .word dodb4
OF  6120 ;
EF  6130 db4bit =*
CH  6140 ;
OO  6150 .byte $88
HL  6160 .byte %11111111
NL  6170 .byte %11110111
GM  6180 .byte %11110011
IM  6190 .byte %10000001
CN  6200 .byte %10000001
EO  6210 .byte %11110011
PO  6220 .byte %11110111
NP  6230 .byte %11111111
GN  6240 ;
PF  6250 dodb4 =*
PL  6260 lda #$84
AB  6270 sta sysdb
IF  6280 jmp clswin
IA  6290 ;
HK  6300 eaddr =*
CI  6310 .end
```

The Lt. Kernal Hard Drive System

Pushing the limits...

by Bill Brier

Recently, several third party manufacturers have released hard drives for use with the C64 and C128. All of these units have their good (and bad) features, but only one is capable of performing in a manner suitable for professional and business use: the Xetec Lt. Kernal hard disk subsystem.

Adapting a hard disk unit to any eight bit Commodore computer is no trivial matter. Both the Commodore DOS and serial data bus are unique to Commodore. The Commodore DOS is file-oriented rather than system-oriented and is relatively unfriendly to first-time users. Also, Commodore drives are intelligent. This means that the host computer has no facilities for running a DOS as would a CP/M or MS-DOS machine.

Lloyd Sponenburgh and Roy Southwick of Fiscal Information, Inc. (a turnkey systems vendor in Daytona Beach, Florida) were well aware of these facts when they decided several years ago to adapt a hard disk to the C64. The result was the original Lt. Kernal hard disk subsystem, which is now assembled and marketed by Xetec Inc. (Salina, Kansas) of Super-Graphix printer interface fame.

Their success in this adaptation results in a system offering capabilities that are normally available only on powerful multi-user mini-computers. The Xetec Lt. Kernal is not perfect but it is far superior to anything else available.

The Lt. Kernal concept

The Lt. Kernal hard disk subsystem is a combination of a small computer system interface (SCSI, pronounced "scuzzy") 5.25 inch hard disk assembly, various interface electronics and a sophisticated user-friendly DOS. The standard capacity is 20 megabytes and this may be increased to 180 megabytes. Additional hardware enables it to multiplex up to 16 computers onto a single drive, resulting in an economical and powerful multi-user system.

The Lt. Kernal implements a modified version of the C64/C128 Kernal. The Lt. Kernal's operating system adds the functions needed to make the host computer "talk" to the hard drive. In addition, the Lt. Kernal DOS adds a variety of immediate mode and program mode commands for file management, directory handling and disk housekeeping. Other hard drives only implement standard CBM DOS commands and do not include the commands that are essential for convenient operation.

The Lt. Kernal DOS and the technology in the drive are the result of the efforts of Fiscal Information, who also own the rights to the name. They support the DOS and the drive technology. They do not actually build or market any Lt. Kernal hardware. The design, assembly, testing and marketing of the finished product are handled by Xetec Inc. They support the users as well as build, sell and service the drive system.

Both Fiscal and Xetec operate bulletin boards for the use of Lt. Kernal owners. On these boards one may discuss various drive topics with Fiscal or Xetec personnel, or receive up-to-the-minute news about new DOS features and improvements.

The Lt. Kernal hardware

A single station Lt. Kernal system consists of the hard disk assembly, a cartridge (the host adaptor), several jumper leads, an interconnecting cable, user's manual and a floppy disk with the Lt. Kernal DOS. C128s also require the internal installation of an MMU daughter-board assembly. The host adaptor is computer powered while the drive has its own separate power source. The Lt. Kernal hardware is designed for continuous operation.

A multi-user system will also require a host adaptor and cable for each computer (and the daughter-board if it's a C128) and one or more multiplexers. A multiplexer can accept four stations, with additional stations (up to 16) being accommodated by daisy-chaining more multiplexers. A multi-user system may be a mixture of C64s and C128s.

The Lt. Kernal hardware is well designed; attractive and professional in appearance. The drive is in a low, flat metal case about the size of two 1541s placed side by side. The on/off switch in the back is the only user control. The unit's modest appearance belies the power and versatility within. A "busy" LED indicates data access. I would like to see a power-on LED as well, as the noise from the drive is barely audible. The only sound is a faint hum from the Seagate 5.25" Winchester drive unit and a soft whirring sound from the fan.

The host adaptor in the cartridge port has access to the system address and data bus lines. However, the adaptor doesn't extend the port. The host adaptor is enclosed in a metal case for maximum shielding and has four rubber feet. The DB-25 receptacle on the back, which connects it into the system bus is directly anchored to the steel chassis and is not at all fragile. A pushbutton marked I.C.Q.U.B. (Image-Capturing Quick Utility Backup) is the only visible control. This is the Lt. Kernal equivalent of an ISEPIK or CAPTURE cartridge and functions in C64 mode only (as of this writing). As received from Xetec, the host adaptor is visible in the $DF00 I/O block of processor address space as a multi-port device. To change the adaptor address to the $DE00 range, simply relocate a jumper on the host adaptor board.

Inside is a four-position DIP switch which is part of the multi-user system arrangement. On a multi-user system, each computer has a station or port number. The port number is determined by the setting of this DIP switch and is displayed as part of the Lt. Kernal prompt. On a single station system, the DIP switch is set to 0 (port numbers range from 0 to 15 inclusive). In a multiplexed system, station 0 becomes the "master" station. Additional stations are set to other port numbers and are designated as "slave" stations.

The port number at location $DE04 (or $DF04, depending on the I/O block chosen) can be read with: lda $DF04 ($DE04) and #%00001111). It is possible for multi-user systems to embody software features that are contingent on which station is being used.

The host adaptor's parallel DMA interface operates at tremendous speed. It is this feature which makes the Lt. Kernal the best choice for business and professional use. Other drives use either the serial or IEEE-488 bus. There is no contest when it comes to speed comparisons, as we'll see below.

Installation of an MMU daughter-board requires that the C128 be opened, the MMU removed from its socket, the daughter-board plugged into the MMU socket and the MMU itself plugged into the daughter-board. An additional modification must be made to the C128 to accommodate the serial port burst mode functions. Although this may sound difficult, the manual gives clear instructions and drawings and the results are certainly worth the effort.

A 25-conductor cable connects the host adaptor to the drive or multiplexer. This cable is of high quality and is designed for maximum shielding to avoid interference problems. Although the supplied cable is relatively short, it is possible to extend the bus a considerable distance if required. There are no user controls on the multiplexer (which is also in a sturdy metal case) and therefore it may be located in an out-of-the way place.

The floppy disk supplied with the drive contains the entire Lt. Kernal DOS (which is already installed on the drive when Xetec ships it). The DOS is serial number matched to the drive as a means of guarding against installing the wrong DOS on the drive (different DOS packages are used for different sized drives). Unlike Commodore DOS, the Lt. Kernal DOS is software and therefore may be upgraded when necessary. By supplying it on floppy disk rather than on a ROM chip costs are reduced and an inexpensive and convenient means of supporting older drives is established. A process referred to as SYSGEN (SYStem reGENeration) allows a user to upgrade or repair the DOS easily.

The Lt. Kernal software

The superior hardware features of the Xetec Lt. Kernal are complemented by a powerful and user-friendly DOS. The Lt. Kernal DOS is executed in RAM in the host adaptor and offers many new immediate mode commands. This amounts to a major overhaul of the computer's operating system and user interface and gives rise to concerns about compatibility with the host computer and the software that is to be used with it.

Fear not, gentle reader! With a few exceptions, the Lt. Kernal DOS peacefully co-exists with any software that has been properly written (that is to say, uses the Kernal jump table and does not JSR directly into ROM routines). Commodore DOS commands are supported (with a few exceptions) and all file types are implemented, including RELative files. C128s equipped with the Lt. Kernal function equally well in C64, C128 or CP/M modes. Whole-drive formatting is not allowed and there are no file-level direct access commands (such as U1: or U2:), these being intentionally omitted to protect the disk-resident DOS (there are undocumented low-level system calls that may be used to read or write any sector on the drive).

The Lt. Kernal DOS offers these safety features and a bevy of new commands - sort and print directories; find a file's load address; copy large groups of files from one drive location to another; recover accidentally deleted files; list a BASIC program to screen non-destructively; read SEQ files; group files into a separate area on the drive; change device number; auto-execute a program on power-up (from either C64 or C128 mode). All that and more is available, making the Lt. Kernal a joy to work with.

The Lt. Kernal supports partitioning (sectioning) of the drive into user-definable areas. Partitioning on a hard drive is an essential feature for serious use, as literally tens of thousands of files may be stored. The Lt. Kernal DOS allows the definition of up to 11 logical units (0 to 10 inclusive). LU 10 is reserved for the DOS and various utilities supplied with the system. The user may reserve space for LUs 0 through 9 and may also store files on LU 10 (space permitting). Each user-definable LU may be configured as a CBM LU or CP/M LU. Any LU may contain up to 4,000 directory entries. In theory, a drive with 11 defined LUs could store 44,000 files.

In immediate mode an LU is selected by lu n <RETURN>, where n is the LU number. In a program an LU may be specified in the syntax of a standard CBM DOS command. To open a

file on LU 6 you would use the syntax: open2,8,2,"6:filename". Neat, huh? It is also possible to select an LU via the command channel. As with Commodore drives the Lt. Kernal command channel is channel 15.

Each LU may be divided into a maximum of 16 user areas (sub-directories). A user area is selected by user n <RETURN> or via the command channel when in program mode. Once logged into an LU and user area, most disk activity will be restricted to that area. Files may be assigned to a given user area by logging into that area before saving or by including the LU and/or user number in the file save syntax. You can move or copy a file from one user area to another as well.

Once logged into an LU and user area, the dir command allows pattern-matching with both leading and trailing "don't cares", direct output to printer, alphanumeric sorting of filenames before output, selective display of file types, viewing of filenames from foreign areas (i.e. LUs and/or user areas other than the one currently logged) and more.

A directory display includes: filename; size in disk sectors (512 bytes); file type (a numeric code that distinguishes ML programs from BASIC, among others); file's load address; the file's physical location within the LU (displayed as a hex address); file's assigned LU and the status of the file's "dirty" flag. (The dirty flag indicates whether the file has been modified since the last archiving operation.) In a C128 in 80 column mode, the directory is neatly arranged in two columns.

Using the Lt. Kernal

We're not talking about a simple plug-in accessory. This is a whole new operating system and programming environment for the C64 or C128. The drive implements high speed, high storage capacity, a fool-proof DOS and ease of use.

The parallel bus interface of the Lt. Kernal results in immediate response and superb performance during loads or saves. Programs are running in an eye-blink and saves occur at as rapid a rate. Also, the nasty SAVE@ bug does not exist on the Lt. Kernal.

At 1MHz (computer speed), the Lt. Kernal transfers data at 38K per second, over 100 times faster than an unmodified 1541 drive. On a C128 at 2MHz (FAST mode), the transfer rate is increased to over 60K per second - about 12 times faster than a 1571 or 1581 in burst mode and over 50 times faster than an IEEE unit interfaced through the cartridge port. Testing has shown that a C128 in FAST mode can fetch a disk sector (512 bytes) into computer RAM in as little as 10 milliseconds. Sector writes are just as fast. Again, there is no contest when it comes to speed comparisons.

The "latency" of the Seagate (the time required for a given sector to pass under the head) averages 8.3 milliseconds, whereas the SFD-1001 averages 100 MS. The lower the latency, the faster the data may be read or written. Additional gains are achieved by extremely dense storage on the media and by the use of multiple read/write heads. This reduces the number of seeks required to read or write a sector and substantially improves performance. Continued research on hard disk design has improved reliability and speed while reducing cost and physical size. These improvements are evident in the technology of the Lt. Kernal. In a year of continuous use, my 20MB unit has been trouble-free.

Inherent speed aside, credit must also be given to the DMA interface and the Lt. Kernal DOS. If the drive had been interfaced via a serial or IEEE bus and if the standard CBM DOS had been utilized, the drive would have been little faster than the floppy units it was designed to replace.

User-friendly DOS

The new functions implemented by the DOS are easy to use and immediate in action. Plain language prompts and error responses guide you through most tasks, making for an intuitive operating environment. Immediate mode command syntax is generally quite obvious, and easier to remember than the equivalent CBM commands.

For example, type "1 filename" <RETURN> to load a file instead of dload "filename" or load "0:filename",8. "L" will automatically load a file to its correct address, with an additional distinction being made if the file is BASIC rather than machine language. Entering "1 2:3: filename" <RETURN> loads filename from LU 2 USER 3. This allows you to load across USER and/or LU boundaries. Within a program, standard CBM commands are used and standard CBM disk error messages are generated. This means that most software will run on the Lt. Kernal without alteration, assuming that it was written to use the standard Kernal jump table.

Specialized DOS functions (such as multiple file deletes) utilize status messages and confirmation prompts, especially if potentially destructive. For example, activating an LU produces the same result within the LU as formatting a disk does on a CBM drive. Because an inadvertent activation could destroy thousands of files, a triple confirmation system is used to protect the user from himself.

A single file may deleted from immediate mode with the "era" (erase) command. Era may be used across LU and/or USER boundaries and there is no confirmation prompt. Era may be used with a pattern-matched filename but the command will scratch only the first file found to match. Type "oops" <RETURN> immediately after an errant scratch and the drive will recover the file.

Multiple file removal may be accomplished with the autodel command. The drive will request the source LU and USER area and list those files on the screen. Using the cursor keys and the space bar, you select the files to be deleted and then tell the system to do its job. Multiple confirmations protect you from careless typing.

Upon powering up the computer, the Commodore sign-on message appears and the Lt. Kernal performs a diagnostic test of the hardware and DOS. When all is well, the Lt. Kernal prompt will appear, indicating: 64 or 128 mode, current LU and user area, and the port number of the station. The Lt. Kernal will search the power-on LU for a program called AUTOSTART and, if found, run it. If AUTOSTART is not found, control is passed to BASIC. This whole process takes perhaps five seconds.

Who needs the Lt. Kernal?

If you write a lot of software, or use the computer for business or other professional use, then the Lt. Kernal is the drive for you. For the professional programmer or the business user, the Lt. Kernal means greater productivity as well as a more reliable and efficient medium upon which to store and retrieve data. For the BBS sysop, the Lt. Kernal means lots of space for uploads and user messages.

The utility of the Lt. Kernal is significantly enhanced if new software is written to take advantage of the special features - the multi-user capabilities, for example. A proficient programmer can write software that allows file sharing amongst the various stations, resulting in greater system utilization.

Another special feature is the implementation of a unique (to Commodore-based systems) file type: the KEY-INDEX file. The KEY-INDEX file may be used to relate data keys to the records of a RELative file or random access storage system. The KEY-INDEX file is controlled by the DOS's KEY file processor, which may be used by BASIC or ML programs. The program simply passes the key string, its record number and some instructions to the KEY file processor and the Lt. Kernal does the rest. The DOS passes back information to your program on the success of the operation and so forth.

KEY file operations are very rapid. A single key and its record number can be retrieved from literally thousands of keys in less than 100 milliseconds. Keys are always inserted into the index in alphanumeric order, key duplication not being allowed. Writing a database to utilize a KEY-INDEX file means that you don't need to devise search and sort subroutines to do the housekeeping. The KEY file processor does it all for you.

Using simple techniques, you can retrieve keys in ascending or descending order or on exact match. When a key is located, the associated record number is retrieved for access to a companion RELative file. In fact, a KEY file may have multiple directories, such a KEY file being the equivalent of a multi-dimensional RAM data array. This is indeed a database programmer's dream come true. The KEY-INDEX file makes a RAM-based index as outmoded as a vacuum tube mainframe.

Complementary to the KEY-INDEX file structure is a greatly enhanced RELative file implementation. On the Lt. Kernal, RELative file record length may be up to 3,072 bytes with a maximum of 65,535 records per file. The maximum possible size of any given RELative file is 16.78 megabytes. Record position commands are executed much faster than on CBM drives and a double-position dance is not required for reliable performance.

There are numerous other features embodied in the Lt. Kernal hardware and DOS, a discussion of which would fill another whole article. However, this is not supposed to be a sugar-coated hardware review. It is always easy to emphasize the good features over the not-so-good and therefore I'd like to mention those features that I don't consider to be optimum.

It's a great system but...

The Lt. Kernal comes with a manual that has been printed and bound in the same manner as the manuals supplied with expensive MS-DOS software. However, the manual is far from complete and will prove to be heavy reading for the neophyte. Although the manual thoroughly describes the installation of the drive hardware and documents the Lt. Kernal DOS commands, it glosses over such hard drive concepts as logical units, subdirectories and how the DOS operates. A quick command summary card is included but it does not shed any more light on the workings of the DOS than can be found in the manual text itself. If you purchase a Lt. Kernal system be prepared to do some experimenting with commands. For example, the manual doesn't mention that reading a directory from within a program will return only the directory of the currently logged user area. Nor does it mention that immediate mode DOS commands are ignored unless the typed command starts at the left margin of the screen.

According to Lloyd Sponenburgh of Fiscal Information, an improved manual and a "power users' kit" are in the works. Presumably, the power users' kit will document low-level DOS calls for advanced programming applications and will describe the inner workings of the DOS in greater detail. Such knowledge will be essential if you ever intend to write a multi-user software package or wish to make full use of the drive's speed and power.

There are some less than optimum conditions in the combination of drive, DOS and computer. The Lt. Kernal DOS constantly monitors system activity to determine if a Lt. Kernal DOS command has been issued or if a call has been made to the CBM Kernal subroutines responsible for peripheral activity (such as CHKIN, CHKOUT and so forth). If it detects disk-related activity, it temporarily remaps the system, causing certain DOS routines to appear in place of some areas of RAM. This is the primary means by which user or program DOS commands are intercepted and serviced. This takes time and, in some circumstances, reduces the computer's operating speed.

A reduction in processing speed will be evident in any function that uses the Kernal BASIN, GETIN or BSOUT subroutines. This effect will be quite noticeable when using the RS-232 routines at 1200 or 2400 baud or when running a C128 in SLOW mode. The Lt. Kernal's presence has a greater effect on the C128 because of its banked memory environment. This,

coupled with the greater complexity of many C128 I/O routines, simply means slower operation (only so much can be done with an eight bit CPU). Needless to say, the slower operation under the Lt. Kernal DOS is less of a problem with the 128 in FAST mode and is less noticeable in BASIC programs than in ML or compiled BASIC programs.

Because of the interception of the BASIN and BSOUT subroutines, SEQuential and RELative file access is actually slower than the IEEE drives. This is less a fault of the Lt. Kernal than of the CBM Kernal itself, as many redundant checks are performed when the BASIN or BSOUT subroutines are utilized. This intensive activity, coupled with the extra code required to pass data between computer and drive slows down the system. Improvements to this section of code are being implemented in the next version of the Lt. Kernal DOS and that BASIN and BSOUT will perform at a much higher speed.

With one exception, the Lt. Kernal DOS operates transparently as long as the programmer uses the CBM Kernal jump table and does not JSR directly into I/O routines in ROM (which is bad programming practice). The exception is that the low-level or "primitive" Kernal I/O calls (TALK, LISTEN, etc.) are not supported. Any calls to the primitives will be sent directly to the serial port. This means that when running in C64 mode you can forget about using the DOS Wedge to issue commands to the Lt. Kernal (which would be pointless anyhow). However, the Wedge load and save commands will work with the Lt. Kernal and any commands prefixed with the @ symbol will be passed to the serial port. Therefore, you may use the Wedge to control a serial port floppy drive that is also connected to the system.

In C128 mode, all BASIC 7.0 DOS commands are supported except HEADER and COLLECT (neither of which has any purpose on the Lt. Kernal). As mentioned before, the DIR command permits the direct output of the directory to the printer (without pagination). Also, it appears that DOS doesn't verify that the printer is on-line, as I've had the system crash when attempting to print to a non-existent printer.

Because of the memory limitations of the C64, the Lt. Kernal DOS swaps the $C000-$CFFF range of RAM out of processor space when certain immediate mode commands are utilized. Upon completion of the command, the contents of this range are restored. This won't present a problem unless you have an interrupt-driven routine in this area. For example, if you request a directory from the Lt. Kernal, the $C000 block will temporarily become part of the DOS. If an interrupt is directed to this area of RAM the machine will probably crash - the IRQ will not find the appropriate code, but will instead see Lt. Kernal DOS code. The same limitation holds true for several other Lt. Kernal utilities. It seems to me that this problem could be avoided by stashing the current page three indirect Kernal vectors on the drive (where there's lots of room for such activity), temporarily resetting all of the vectors to their default values and then restoring them to their original condition once the processing has been completed. As it is you must exercise care to avoid system fatality. For the non-technical user this may represent a source of frustration and may lead him or her to believe that there is something amiss with the drive.

With one exception, no memory usage restrictions appear to exist in C128 operation. The exception has to do with the use of the I/O block at $DF00. The STASH, FETCH and SWAP statements in BASIC, the DMA-CALL subroutine in the Kernal, and CP/M (when using drive M) all address this area, as this is where the external RAM expander is mapped into the system. To use the RAM expander or to run CP/M, you must move the I/O page jumper on the host adaptor so as to map the adaptor into the $DE00 block. This may prevent protected C64 programs captured with I.C.Q.U.B. from functioning.

In terms of software compatibility, a few problems may arise. Any database program that utilizes direct-access storage and retrieval methods (U1: or U2:) is not going to operate with the Lt. Kernal. This means that older versions of Superbase will not operate (the more recent version that uses RELative files will work). Most database managers, word processors and spreadsheets will operate if they utilize standard CBM file types. Needless to say, any software that is dependent upon the inner workings of the 1541 DOS (such as applications that set up some kind of speed-up function in the drive) are not going to run. Programs that rely on the internal timing of the 1541 ROMs or attempt to utilize low-level DOS functions will go belly up. I.C.Q.U.B. functions only in C64 mode as of this writing so C128 software that has been protected by DOS protection schemes cannot be transferred to the Lt. Kernal. To utilize such software with the Lt. Kernal you must change the drive's device number (a simple immediate mode command) and load the software from the floppy drive.

One other compatibility problem exists that may be important if you wish to use KEY files with database software written in BASIC. The BASIC syntax for manipulating KEY files is not compatible with any of the BASIC compilers that are presently available. This is because a colon is used to separate the SYS call to the KEY file processor from the list of variables that is associated with the call. Most compilers can be instructed to ignore a program line fragment by placing a double colon (::) before the fragment, the result being that it will be passed directly to the BASIC interpreter. Compilation will then resume at the next colon or at the start of the next line. However, the colon following the SYS call to the KEY file processor will tell the compiler to attempt to compile the list of variables that follows the SYS call. The compiler will then flag the list as a syntax error. This is unfortunate, as a compiled BASIC database using a KEY file would make a very nice and efficient package.

If there is one significant weakness in the Lt. Kernal system, it is the means by which data backup is performed. Any data loss on a hard disk system could be massive. To ensure data security, frequent backups are mandatory. Unfortunately, the only backup method presently available to a Lt. Kernal user is

continued on p. 73

The 1351 Mouse and GEOS 1.3

Graphic environment on a roll

Review by Malcolm O'Brien

The 1351 mouse was well worth the wait. What a gas! What a great product! With its sleek and attractive styling (identical to the Amiga mouse), the 1351 mouse is a perfect complement to your 64 or 128. It has a very solid feel and, to my hand, a more ergonomic design than the mice you'll find attached to Lisas, Macintoshes or PCs. I particularly like the tactile feedback on the two buttons.

Two Modes

The people at Commodore have cleverly given the 1351 mouse a dual personality. It has two modes of operation, selectable on power-up. With the mouse plugged in, hold down the right mouse button while you turn on your computer. Now your mouse will be disguised as a joystick and will function properly with any software that expects to find a joystick. Actually, this disguise is more like the 1350 mouse, the joystick in mouse clothing. It should be noted here that some users have reported that mice make lousy joysticks. Certainly, this is not the way to have a rip-roaring game of Screen Busters from Outer Space, but it may be just the ticket in a different sort of application; for example: hi-res drawing programs like Doodle, sprite editors or font editors. You may also find it suitable for non-arcade type games like Shanghai. Experiment!

If you power-up without holding down the right button, the mouse will be initialized as a true proportional mouse. It is in this mode that the 1351 mouse is in its glory and really offers Commodore users something new.

Documentation

The documentation is up to Commodore's usual (new) standard: very good! A small booklet included with the package contains a short section on using and caring for (but not feeding) your mouse. There is one small discrepancy here between what the booklet says and the way things are in the real world. The booklet advises cleaning the mouse's metal rollers with alcohol or head-cleaning fluid on a cotton swab. On disassembly, however, it will be seen that the rollers are actually plastic cylinders on metal spindles. Note that you should *never* use solvents like alcohol or head-cleaning fluid on these plastic parts. Keep your mouse clean by ensuring that you always use it on a clean surface. Even so, a periodic dusting is recommended. Just disassemble your mouse as instructed, wiping the ball with a soft cloth and blowing into the opening.

Programming

The second section of the booklet is longer and offers an in-depth discussion of mouse internals for those interested in offering mouse support in their own programs. In joystick mode, this is fairly simple - it's the same as programming for a joystick with one small (and generally ignorable) exception. When the 1351 mouse is functioning as a joystick, the left button serves as the fire button in the standard way. However, the right button is readable. It's mapped into the SID POTX register. When the right button is pressed, the register will contain a value less than $80. When the button is not depressed, SID POTX will contain a value greater than or equal to $80. I call this an ignorable feature since it is not a joystick function. If your program is going to read the right button, the operator won't be able use this function if he or she is using a joystick. (As an aside to the readership: What is the right button for? If GEOS uses it, I don't know how. Anyone else?)

Programming the 1351 mouse in proportional mode is an entirely different kettle of fish. This is not a simple task, especially the positioning aspect (the left and right buttons appear as joystick lines). If you're not into machine language, or are intimidated by phrases like: "wedge into the IRQ handler prior to the polled keyscan" or "distinguish between a point short in the keyboard matrix and a whole row or column being grounded", then you will have a lot of difficulty programming the mouse yourself. There is an alternative however...

The Disk

Of course, the best hardware is nothing more than a pricey doorstop without software. Included in the 1351 mouse package is a disk of the 'flippy' persuasion. Side A has several demo programs for the 64 or 128 (in native mode). These include: mouse drivers in assembly source, BASIC loader and raw machine language. Also included is a simple "Identify the Shape" educational program that serves as an example for writing BASIC programs that get mouse event data from the ML drivers. This technique serves to make even a simple BASIC program look more sophisticated and professional.

At present there is very little commercial software available that will make use of the 1351 mouse (at least in proportional mode). Obviously, most mouse users will be using the device with GEOS and will need no other justification for their purchase. The only other commercial software that I'm aware of that offers support for the 1351 mouse is CADPAK from Abacus Software. There may be other products but I haven't seen them yet. Nor have I used CADPAK, although it would definitely seem to be an appropriate application for this device.

GEOS V1.3

Side B of the included disk has only one file. This is the GEOS upgrade to Version 1.3. Note that you cannot use the 1351 mouse (in proportional mode) with Version 1.2 or earlier. Although the upgrade program is copy-protected, it may be freely re-used to update anyone's GEOS system disk, and it should be so used. Upgrading is a good idea even if you're not using the 1351 mouse. The new version is changed in several important ways: new printer drivers, new input drivers, new utilities, safeguards and shortcuts.

First, the new input drivers: the Flexidraw lightpen and the Koala Pad. You can switch from joystick to mouse to pad to pen without rebooting with "select input" under the GEOS menu. Note that the pad and pen cannot use the scroll arrows in geoPaint. Use the page position indicator at the bottom.

The utilities: Backup, Disk Copy, Configure and Rboot. Backup is now only for use with the GEOS system disk. Use Disk Copy for copying work disks. Configure allows the use of a RAM expansion unit. You can create a RAM 1541, 'shadow' a real 1541, use DMA for fast data transfers, and enable fast rebooting. If the deskTop is in RAM, tapping the RESTORE key will reboot GEOS from RAM - fast!

The safeguards: deskTop 1.3 and Disk Copy will not allow you to screw up your Master disks. You won't be able to format them or use them as "destination" disks. Nor will you be able to delete important files or even relocate certain files. This is going to spare a lot of users "that sinking feeling..." One extra safety note, though. You can't use the deskTop 1.2 or the old Preference Manager with the 1.3 GEOS Kernal. To do so is to court a crash (speaking from experience here!).

Finally, the shortcuts: These are keystroke combinations for functions that used to be menu-only. Shortcuts are accessed by holding down the logo key and pressing another key. The deskTop has three: Logo-I allows you to select a new input driver. Logo-O opens a disk and logo-C closes it. geoWrite has numerous shortcuts, which are shown in the menus.

The geoPaint update "handles text scraps better" according to Berkeley Softworks, and forces the edit box to conform to colour card boundaries when working in colour mode.

You get a lot for your money in this package and it's all great! I love mine and you'll probably love yours too.

Lt. Kernal... continued from p. 71

to copy files from the hard drive to a floppy disk drive. According to Lloyd Sponenburgh, a cartridge-type IEEE interface may be used to connect an IEEE drive. With a 1541 you will need 118 (that's not a misprint) floppy disks to back up your 20MB hard drive - assuming that the drive is full). With a 1571, or if you use both sides of the disks on a 1541, you will need 59 floppies. A 1581 user can manage with a mere 25 disks while an SFD-1001 user will be able to get by with only 20 disks. What makes this backup method especially onerous is the fact that the only proper way to back up a high capacity drive is the "double grandfathering" method. This requires the use of two complete sets of disks, thus protecting you in the event of a major system fatality while performing a backup.

Regardless of the drive used, backups will be time consuming. If you have a 1541 or 1571 drive, the built-in FASTCOPY utility will allow a copy to be cranked out once every three minutes or so (FASTCOPY runs only in C64 mode). A little math will tell you how many hours you'll need to perform a full backup. FASTCOPY reprograms the floppy drive to speed up copying. Therefore, it is unlikely that it will function with a 1541 clone (it wouldn't operate with my MSD SD-2). [For what it's worth, the FSD should work in this case. - Ed.] In such a case, or if you are using an IEEE drive, you can use 'copy-all 64" (supplied on LU 10 of the Lt. Kernal) or "uni-copy'. Neither of these copiers speeds up the serial bus.

Unfortunately, there is no mechanism presently available to copy a Lt. Kernal KEY-INDEX file to or from a floppy disk. For a business or other professional user, the backup situation represents a significant limitation. Most businesses simply cannot afford the time required for a full backup. Yet a business cannot afford to not back up the drive. Although FASTCOPY lets the user selectively back up only the most recently modified files, he would still be faced with a daunting task. One solutionwould be a high-speed streaming tape backup. A ttape streamer can back-up 20MB in under 10 minutes. Xetec has done some work in this area but, as of this writing, has not released any hardware.

To buy or not to buy...

At approximately $900 (US), the price is not trivial. However, for a major breakthrough in high capacity mass storage in terms of features and ease of use, it's a great value. Consider: two SFD-1001's, an IEEE interface and cables will cost approximately $600 (US) and will only give 2.1 MB, 1.2K/second speed and no DOS enhancements. My only reservation in recommending the Lt. Kernal for business or professional use is the backup situation. A better system is urgently needed if the Lt. Kernal is to make its mark in the business world. However, if you can live with the present backup scheme then the Lt. Kernal is definitely the way to go. The Lt. Kernal is not perfect but it is close! And, it is constantly being improved.

Contact Xetec, Inc. at 913-827-0685 for more information.

Warp Speed

"Impulse power is not enough, Mr. Scott"

Review by Malcolm O'Brien

Warp Speed is one of the newest entries in the DOS enhancement sweepstakes and stands poised to become a front runner. Warp Speed is powerful, flexible and easy to use. A reset button is built into the cartridge, along with a 64/128 slider switch. Warp Speed will appeal to a broad base of users due to the number of devices supported. Warp Speed works with: the 64, the 128 in native mode (40 or 80), the 1541, 1571, 1581, MSD (!) and some hard drives. An extended DOS wedge is included with support for multiple drive systems. All features are accessible from menus to make things simple for new users while the long-time hacker can bypass the menus in most cases and use one or two keystrokes to initiate the magic.

Warp Speed is easier to use than it is to document. It has so many features that describing them all results in a long review. It's great to have this kind of power at your command. But it wasn't always this way...

A little background

The C64 and 1541 seemed like a step backwards to PET users who had BASIC 4.0 disk commands and quick, parallel dual drives such as the 4040. At that time the obvious path for drive enhancement was to interface the C64 with the faster IEEE disk drives. Many users (including me) are still using IEEE drives via G-Links, BusCards etc. (To be fair, it must be noted that the introduction of the serial bus interface did help to keep the hardware costs down.)

As the flood of C64 software turned into a tidal wave, more and more commercial (read: copy-protected) programs relied on 1541-specific drive ROMs. Another step backwards - we now needed to use 1541s to be able to use some software. And so it was that the C64 community was offered Kwik Load, Fast Load, Vorpal, SuperDos, GT-4, Mach and others. You probably have one (or more) of these yourself.

Fast Load may have been the most popular of these. Even now, years later, Fast Load is still prominently displayed in every computer store I browse and, presumably, is still selling well. It was an effective solution for the problems described above but added new problems of its own design (skewed directories principally). In spite of this, it was parked in my cartridge port for three years or so.

But not any more. Warp Speed is how I spell relief now. Warp Speed has powers and abilities far beyond those of mortal cartridges. It's clearly superior to Fast Load and is well worth the difference in price (about $10 here in Toronto).

What the user will find

First and foremost, the speed increase is not just in the loads. Saving and verifying also happen at Warp Speed. (Tech note: Files saved with Warp Speed are saved in a "skew 6" format. These files will warp load ten times faster than normal 1541 speed.) The DOS wedge includes a quick text file reader, the ability to set the currently logged drive and single-key entry to the menu system (British pound key) or the machine language monitor (pi key).

The text reader is a nice addition. Just type an ampersand (&) followed by the name of the text file and hit Return. The screen clears and the text begins to be printed to the screen. CTRL may not slow it down enough for reading so use the spacebar to pause and restart the listing. RUN/STOP will exit. This is similar to the TYPE command in MS-DOS and CP/M. It's great for reading files or just to take a quick peek to determine a file's contents. I use this feature a lot and you probably will too.

Setting the currently logged drive is also common to the MS-DOS and CP/M environments. This allows you to leave out the ",8" or ",9" when accessing the drive. To switch between the two, type a number sign (#) and Return. This will toggle between devices 8 and 9. If you're using more than two drives, follow the number sign with the device number of the drive you want to operate on.

Note that Warp Speed will search both/all drives for the file desired and, if found, will switch the currently logged device to that drive. Commodore-RUN/STOP will always load the first file on the disk, *not* the most recently accessed.

The DOS wedge

As usual with the wedge, you preface a disk command with the at-sign (@) or a "greater than" (>). The at-sign alone will read the error channel. You use a slash for loading BASIC, a

left-arrow for saving BASIC, a percent sign for ML loads and an exclamation point for a verify. An unusual wedge feature is the "f" command. This will yield a fully verified fast format (22 seconds) and even includes an "Are you sure?" prompt.

The non-destructive directory that is initiated by typing a dollar sign followed by a Return can be paused and restarted with the spacebar or aborted with RUN/STOP. All pattern matching and multiple parameters are supported; i.e. "$*=seq" or "$p*,t*,s*" will work properly. Beats me why they never document this stuff!

Utility commands

The other directory function is one of the Utility Commands. All of these begin with an up-arrow. When followed by a "$", the disk auto menu is enabled. This will load in the directory and allow you to scroll through it with the cursor keys. Pressing Return will warp co load the highlighted file and run it. I was pleasantly surprised to discover that if you decide not to load a file and abort the auto menu with the STOP key, your BASIC program is still in memory. Note, however, that if the BASIC program in memory is very large, the directory load will corrupt BASIC.

Here's a quick description of the rest of the Utility Commands (each preceded by an up-arrow):

- **k** - Kill: fast loader only. Other functions are unchanged
- **e** - Enable: resets the Warp Speed load, save and restore vectors
- **u** - Unnew: restores BASIC after a NEW or pressing the reset switch
- **r(n)** - Renumber: assign current drive device number **n** (default is 8 to 9)
- **h** - Hardcopy: dump text screen to printer (upper-case/graphics)
- **s** - Single side: put 1571 into 1541 mode
- **d** - Double side: put 1571 into native mode

Note that both format commands function in accordance with the 1571's current mode.

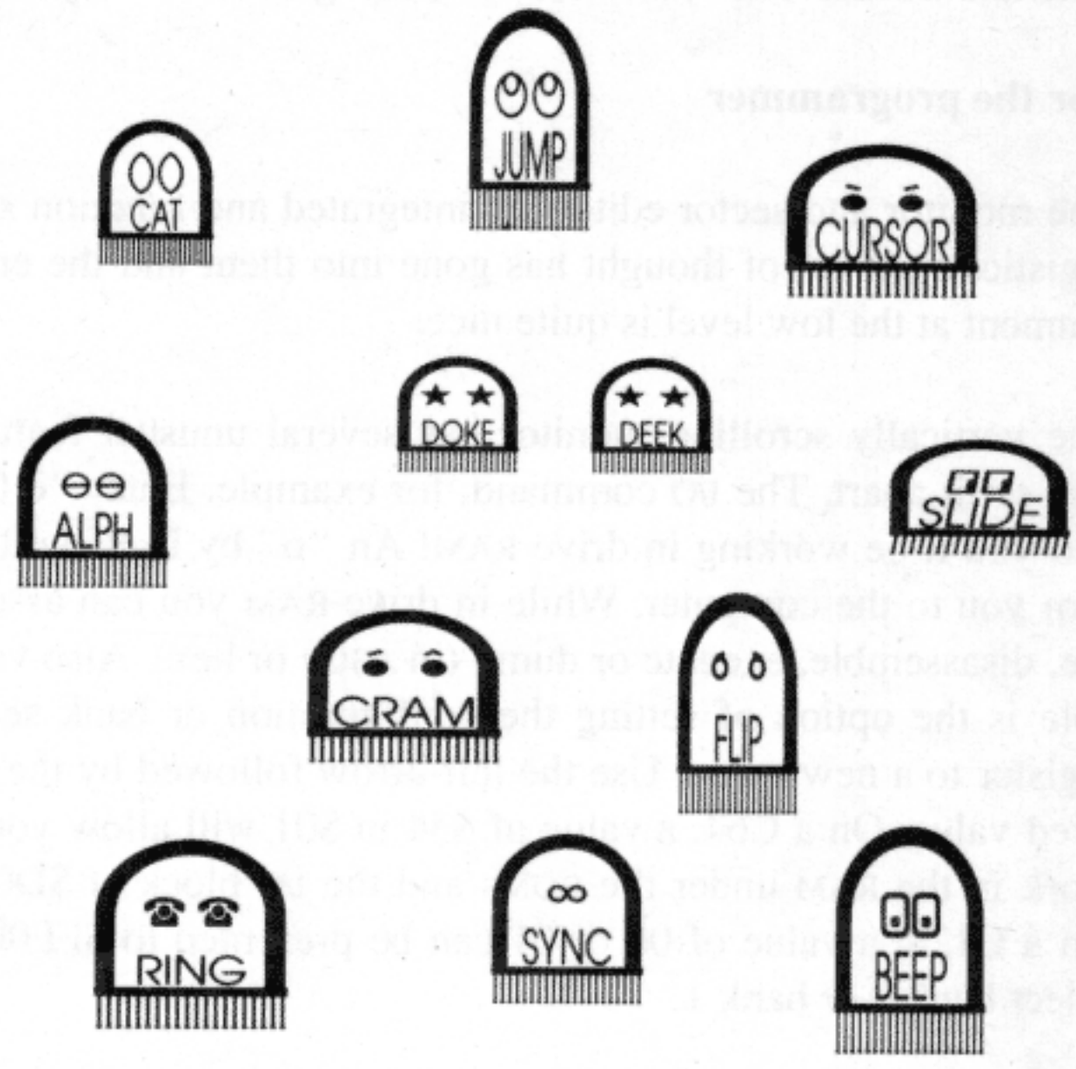
Multi-file/whole disk operations

These operations are selected from the Main Menu which is brought up by entering the British pound key. Selections are made from the menu by number or by cursoring. Functions include single drive copier, two drive-nibble copier and the ability to copy or scratch multiple files. (Typing an "a" will select all files. An "r" will select remaining files below the cursor. Home will move the cursor to the top of the directory. An "s" starts the function when selections are completed. Operation status is indicated throughout.)

The two drive copier will duplicate a single-sided disk in 30 seconds! This copier uses write verification and will report any errors encountered during copying. Although the documenta

New! Improved! TRANSBASIC 2!

with SYMASS™



"I used to be so ashamed of my dull, messy code, but no matter what I tried I just couldn't get rid of those stubborn spaghetti stains!" writes Mrs. Jenny R. of Richmond Hill, Ontario. "Then the Transactor people asked me to try new TransBASIC 2, with Symass®. They explained how TransBASIC 2, with its scores of tiny 'tokens', would get my code looking clean, fast!

"I was sceptical, but I figured there was no harm in giving it a try. Well, all it took was one load and I was convinced! TransBASIC 2 went to work and got my code looking clean as new in seconds! Now I'm telling all my friends to try TransBASIC 2 in *their* machines!"

• • • • • • •

TransBASIC 2, with Symass, the symbolic assembler. Package contains all 12 sets of TransBASIC modules from the magazine, plus full documentation. Make your BASIC programs run faster and better with over 140 added statement and function keywords.

Disk and Manual $17.95 US, $19.95 Cdn.

(see order card at center and News BRK for more info)

TransBASIC 2

"Cleaner code, load after load!"

tion states that this is not as reliable as the fully verified single copier, it has worked perfectly for me every time and is a wonder to behold!

The manual suggests using the single copier if the dual copier should fail. A great feature of the single copier is compression of the read data. You may be able to copy a not-full disk in just one or two passes!

As you are probably beginning to surmise, these functions will allow you to re-organize your disk library with a minimum of time and trouble. And you *do* need to reorganize, don't you?

For the programmer

The monitor and sector editor are integrated and function synergistically. A lot of thought has gone into them and the environment at the low level is quite nice.

The vertically scrolling monitor has several unusual features that set it apart. The I/O command, for example. Enter "o 08" and you'll be working in drive RAM! An "o" by itself will return you to the computer. While in drive RAM you can assemble, disassemble, execute or dump (in ASCII or hex). Also valuable is the option of setting the configuration or bank select register to a new value. Use the left-arrow followed by the desired value. On a C64, a value of $34 in $01 will allow you to work in the RAM under the ROMs and the I/O block at $D000. On a C128, a value of 00 or 01 can be presented to $FF00 to select bank 0 or bank 1.

Another handy feature is the transfer command. This is a smart transfer, i.e. the two blocks of memory can overlap and the transfer "will not turn into an accidental fill command." In addition, you can transfer to and from drive memory with the "td" and "tc" options or toggle output to the printer with the "p" command.

All wedge and utility commands are also available from the monitor. All the other standard monitor commands are included with a couple of variations in their functioning. For example, you can specify an alternate load address when loading or saving a program. A "d" without an end address will disassemble to the end of memory; once again, pause and resume with spacebar, abort with STOP. The hex and ASCII dumps work the same way. Scroll up or down as desired. Overtype an address at the top or bottom of the screen and the monitor will obediently begin displaying from the target memory segment.

Time to leave the monitor now and there are five ways of doing it! The "q" command will exit and restore the break vector to normal, i.e. Commodore's monitor in the 128, warm start in the 64. The "x" command will return to BASIC with the break vector pointing to the cartridge monitor. Switch to the sector editor with "xs" and to the main menu with "xm". The "xc" command will return to BASIC via a cold start which will also clear the break vector. These extra conveniences are part of the reason why Warp Speed is such a joy to use.

The sector editor uses memory from $7E00 to $7EFF as the editing buffer. The default editing mode is hexadecimal but pressing "t" will enable text mode. If you exit to the monitor, the editing buffer and current track and sector values are retained. This allows the option of editing the sector at the opcode level.

Type an "r" to read a sector if the default track and sector is OK; otherwise enter the values in hex. Up and down scrolling will move the cursor through both pages of the sector. Type a "p" if you'd like to dump the block to your printer.

Extra editing features are available while working within a sector. Pressing "SHIFT-CLR/HOME" will fill the buffer with zeros from the current cursor position to the end. HOME will move your cursor to the top of the screen editing area. A second HOME will place the cursor at the top of the sector. From this position, you can get the next sector in the file by typing a "j" which will jump to the track and sector under the cursor. To step through the file from any other position, type an "n" for next. The plus and minus keys will move you one sector forward or back. When used with SHIFT they move you one track forward or back.

Before you write that block back with "w", remember that you have source and destination drives set! If you really want to write back to the source disk, press the spacebar to flip the drive settings. The usual cautions with respect to sector editors apply. Be careful....

Some small problems

The only problems I had while using Warp Speed occurred while using one 1541 and one 1571. I must lay the blame at the rubber feet of the 1571. This is an "old ROM" 1571. The docs for Warp Speed clearly state that you should be using the upgrade ROMs. And you should - even if you're not using Warp Speed. Despite this discrepancy, Warp Speed functioned beautifully with the old ROM 1571 when it was the only drive attached.

I should also mentioned that some software will not fare well with Warp Speed installed. The Q-Link software refused to boot but GEOS disables Warp Speed to use its own turboDisk and you can boot Q-Link from the deskTop. I encountered a different problem while using Sixth Sense on the C128. After a period of time online (full buffer?) I would be dropped into BASIC with garbage characters on the screen. Typing RUN restarted Sixth Sense which then cleared my buffer and hung. On the other hand, the performance improvement with something like SpeedScript is nothing short of remarkable.

All in all, Warp Speed offers much more than fast loading. It's helped a lot in the matter of producing the *Transactor* disk, which requires more work than you would imagine. Users group librarians know something about this too. But the bottom line is that, with its numerous features and great speed, Warp Speed has something for everyone.

News BRK

Transactor News

Submitting News BRK Press Releases

If you have a press release you would like to submit for the News BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue and ones whose intended readership is not clear must unfortunately go straight into the trash bin. We only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance. The News BRK column is compiled solely from press releases and is intended only to disseminate information; we have not necessarily tested the products

Distributors Wanted

Many subscribers state that the magazine is not available in their area. If you know of retailers who are not carrying*Transactor* or *Transactor for the Amiga*, write or e-mail (Compuserve PPN 76703,4243) and send us their names and addresses. We particularly need distributors in: Rhode Island, New Hampshire, Maine, Vermont, Delaware, West Virginia, South Carolina, Alabama, Mississippi, Iowa, South Dakota, North Dakota, Montana, Nebraska, Wyoming, Hawaii, Arkansas, Idaho, Alaska and all over Canada, particularly on the Prairies and in the West. Subscribers and dealers are our most important resource.

The 20/20 Deal

...is still in effect: order 20 subscriptions to the mag or disk, 20 back issues, 20 disks etc., and get a 20% discount. (Offer applies to regular prices and cannot be combined with other specials).

Subscriptions

Please note that your subscription order will run from the *next* issue and cannot be back-dated or our mailing database would freak. This may mean a delay in getting your first issue. If you need back issues, use the order card in the centre of the mag.

No Longer Available

The 1541 Upgrade ROM Kit is sold out. See Volume 7 Issue 2 for complete instructions on obtaining a set; disk #13 contains the ROM image you'll need to burn your own EPROMS. However, we're reasonably sure that the ROM image is compatible with the 1541 only. 1541C owners will need to create an image of their ROM set, then make the changes described in Volume 7, Issue 2, but with minor mods for what are more than likely simple address changes. We are still waiting for an update article from someone who has successfully done this!

"**Moving Pictures**" is no longer available from *Transactor*. If you have ordered a copy, you may ask either for a refund or have a credit issued against further orders from Transactor Publishing - Renanne Turner, our customer service person, will be in touch with you. Moving Pictures is now distributed by CDA, with new packaging and manual. Contact CDA at: P.O. Box 1052, Yreka, CA 96097. Phone (916) 842-3431.

Transactor Mail Order

Items on order cards in back issues of*Transactor* are not necessarily currently available; if you are unsure, please call Renanne before sending in your order. To be certain, place orders from the card in the most recent issue. Please remember that your order takes a week to ten days to reach us. We wil process it as quickly as possible and it will then take another two weeks to reach you by what is alleged to be a Postal Service. If you have a problem, call Renanne (Mondays, Wednesdays, Fridays, 9 AM - 4 PM Eastern time.)

Prices for all products are listed on the order card in the centre of the magazine. Subscribers: you can use the address label from the bag holding your magazine and just stick it on the order card instead of filling it in by hand!

• **Jugg'ler-128** - A product of Herne Datasystems Inc., written by M. Garamszeghy. This program provides read, write and formatting support for more than 130 types of MFM CP/M disks on the C128 in CP/M mode with a 1570, 1571 or 1581 disk drive. It is compatible with all current versions of C128 CP/M and all C128 hardware configurations including the 128-D. All normal CP/M file access commands can be used with the extra disk types. Jugg'ler is available by mail order for $19.95 Canadian or $17.95 US from Transactor. Order from the card at the centre of this magazine.

• **Quick Brown Box** - Battery Backed RAM for C64 or C128. The Quick Brown Box cartridges for the C64/C128 retain files even when the cartridge is unplugged. Unlike EPROM cartridges, the QBB requires no programming or erasing equipment except your computer. Loader programs are supplied and you can store as many programs into the cartridge as its memory will allow. It may even be used as a non-volatile RAM disk. Auto-start programs are supported, such as BBS programs and software monitoring systems that need to reboot after a power failure. All models come with a RESET push button and use low current CMOS RAM powered by a 160 mA-Hr. Lithium cell with an estimated life of 7 to 10 years. Comes with manual; software supplied includes loader utilities and Supermon+64 (by permission of Jim Butterfield); 30-day money back guarantee and a 1 year repair/replacement warranty.

• **The Potpourri Disk** - A C64 product from the software company AHA! (aka Chris Zamara and Nick Sullivan). In-

cludes a wide assortment of 18 programs ranging from games to educational programs to utilities. All programs can be accessed from a main menu or loaded separately. No copy protection is used on the disk, so you can copy the programs you want to your other disks for easy access. Built-in help is available from any program at any time with the touch of a key, so you never need to pick up a manual or exit a program to learn how to use it. Many of the programs on the disk are of a high enough quality that they could be released on their own, but you get all 18 on the Potpourri disk for just $17.95 US/$19.95 Canadian.

• **TransBASIC II** - contains all TB modules ever printed. There are over 140 commands; pick the ones you want to use in any combination. It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

• **Inner Space Anthology** - This is our ever-popular reference book. It has no "reading" material, but in 122 compact pages there are memory maps for five CBM computers, three disk drives and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands and Wordprocessor and Spreadsheet commands. ML codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more ... about 2.5 million characters in total!

• **The Transactor Bits and Pieces Book and Disk** - 246 pages of Bits from *Transactor* Volumes 4 through 6 with a very comprehensive index. Even if you have all those issues, it makes a handy reference - no more flipping through magazines for that one bit that you just know is somewhere. Also, each item if forward/reverse referenced. Bits that are similar in nature or are updates to previous bits are cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing. The bits book disk contains all the programs from the book and can save a lot of typing.

• **The G-Link Interface** - The G-Link is a C 64 to IEEE interface. It allows the 64 to use IEEE peripherals such as the 4040, 8050, 9090, 9060, 2031 and SFD-1001 disk drives, or any IEEE printer, modem or even some Hewlett-Packard and Tektronics equipment like oscilloscopes and spectrum analyzers. The beauty of the G-Link is its "transparency" to the C64 operating system. Some IEEE interfaces for the 64 add BASIC 4.0 commands and other things to the system that can interfere with utilities you might like to install. The G-Link adds nothing: it is so transparent that a switch is used to toggle between serial and IEEE modes, not a linked-in command. Switching from one mode to the other is also possible with a small software routine as described in the documentation.

• **Transactor Disks** - now with their new, colour directory listing labels. As of Disk #19 a modified version of Jim Butterfield's Copy-All is on every disk. It allows file copying from serial to IEEE drives, or vice versa.

• **The Micro-Sleuth: C64/1541 Test Cartridge** - Designed by Brian Steele (a service technician for several southern Ontario Schools), this is a very popular cartridge. The Micro-Sleuth will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine, tests all RAM, ROM and other chips, and in another mode puts up a menu:

1) Check drive speed
2) Check drive alignment
3) 1541 serial test
4) C64 serial test
5) Joystick port 1 test
6) Joystick port 2 test
7) Cassette port test
8) User port test

A second board (included) plugs onto the user port:; it contains 8 LEDs that let you locate the faulty chip. Manual included. Micro-Sleuth with both boards and manual is $99.95 US/$129.95 CDN.

• **Transactor Back Issues and Microfiche** - All Transactors from Volume 4 Issue 1 are available on Microfiche. The strips are the 98 page size compatible with most fiche readers. Some issues are available only on microfiche and are marked as such on the order card. The price is the same as for the magazines with the exception that a complete set (Volumes 4, 5, 6 and 7) will cost just $49.95 US/$59.95 CDN.

This list shows the "themes" of each issue. Theme issues didn't start until Volume 5 Issue 1. Transactor Disk #1 includes all the programs from Volume 4 and Disk #2 includes all programs for Volume 5 Issues 1 to 3. Thereafter there is a separate disk for each issue. Disk #8 from the Languages Issue includes COMAL 0.14, a soft-loaded, slightly scaled down version of the COMAL 2.0 cartridge. Volume 6, Issue 5 lists the directories for Transactor Disks #1 to #9.

• Vol.4 Issues 1 to 3 (Disk #1)	
• Vol.4 Issues 4 to 6 (Disk #1) - MF only	
• Vol.5 Issue 1 - Sound and Graphics	Disk #2
• Vol.5 Issue 2 - Transition to ML (MF only)	#2
• Vol.5 Issue 3 - Piracy and Protection (MF only)	#2
• Vol.5 Issue 4 - Business and Eduction (MF only)	#3
• Vol.5 Issue 5 - Hardware and Peripherals	#4
• Vol.5 Issue 6 - Aids & Utilities	#5
• Vol.6 Issue 1 - More Aids & Utilities	#6
• Vol.6 Issue 2 - Networking & Communications	#7
• Vol.6 Issue 3 - The Languages	#8
• Vol.6 Issue 4 - Implementing the Sciences	#9
• Vol.6 Issue 5 - Hardware & Software Interfacing	#10
• Vol.6 Issue 6 - Real Life Applications	#11
• Vol.7 Issue 1 - ROM/Kernel Routines	#12
• Vol.7 Issue 2 - Games from the Inside Out	#13
• Vol.7 Issue 3 - Programming the Chips	#14
• Vol.7 Issue 4 - Gizmos and Gadgets	#15
• Vol.7 Issue 5 - Languages II	#16
• Vol.7 Issue 6 - Simulations & Modelling	#17
• Vol.8 Issue 1 - Mathematics	#18
• Vol.8 Issue 2 - Operating Systems	#19
• Vol.8 Issue 3 - Feature: Surge Protector	#20
• Vol.8 Issue 4 - Feature: Transactor for the Amiga	#21

- Vol.8 Issue 5 - Feature: Binary Trees#22
- Vol.8 Issue 6 - Feature: Cellular Automata#23

Your Name Here - Mainly due to demand from readers (and we'd also like the money!), *Transactor* is now accepting a limited amount of advertising. If you have a product or service which would be of interest to our readers, you will find the rates a very pleasant surprise. Your advertising dollar will take your message directly to the heart of the Commodore world.

Classified ads are also available at $2.00 per word - we'll do all the typesetting. Either write or phone in your requirements. We reserve the right to refuse advertising which is misleading, fattening or promotes piracy.

Industry News

C128 Developer's Package: Commodore's own C128 Developer's Package for the C64/C128 is suitable for both large and small development projects. The package works best with systems having more than one disk drive and an 80-column text display, but minimal systems are supported as well. The Developer's Pack includes an editor, an assembler, C128 tools, RAM expansion routines, 1351 mouse routines, C64 tools, 1571/1581 burst routines and C64 fast loaders.

The editor, ED128, is a full-screen editor similar in function to the EDT editor from Digital Equipment Corporation. ED128 functions in both ASCII and PETASCII. HCD65 is a powerful 6502 macro assembler similar to the assembler used to assemble the C128 operating system. This assembler supports conditionals, local labels, many directives, cross references, etc. The C64 tools include: a sprite editor, a sound editor, and a character editor. The software is provided on two double-sided diskettes (included).

The manual includes such valuable information as: the differences between the C128 and 1571 ROM revisions; source code for the fast loaders, REU routines, mouse drivers, and burst routines; and descriptions of the routines in the C128 BASIC 7.0 floating point math package including the table of jump vectors. To get a copy of the Developer's Package, order part number CDEV128001 from: CATS, Attn: Lauren Brown, 1200 Wilson Drive, West Chester, PA, USA, 19380.

Complete Bookkeeping Package for the C128: "THE SYSTEM" is a comprehensive, integrated, easy-to-use electronic bookkeeping package for the C-128. The General Ledger, Accounts Receivable Ledger and Accounts Payable Ledger are always up to date; posting is not put off to some future time. In addition, "THE SYSTEM" provides you with a payroll record-keeping function. You are able to print Income Statements which cover from one to twelve months of operation, and go back as far as eighteen months.

"THE SYSTEM" is intended for use as a "point-of-sale" package, actually replacing your cash register. At day's end, a summary of all sales and their cost is printed for each sales clerk and the total for the whole sales force. Other features: analyze performance by sales staff and department; "cash analysis" to assist you in balancing the cash at the end of the day; full purchasing, receiving, and costing capabilities; payments by cash or cheque; complete audit trail; custom-designed statements and reports; intelligent handling of disk errors.

Dataland Ltd., P.O. Box 663, Tottenham, Ontario, Canada, LOG 1WO. Phone (416) 936-2677.

Mystic Jim's Stuff: Mystic Jim's software and hardware are primarily related to GEOS, including products to interface GEOS with other Commodore programs such as Doodle, Koala Pad, Print Shop and BASIC 8 in 80 column mode. Hardware products include a Real-Time Clock and a 64K Video RAM upgrade kit for the C128.

Shareware disks are sent on request. If you find a disk useful, you may request any or all of the others, on the shareware basis: you contribute whatever the disks are worth to you, after trying them. Shareware membership is available for $50 (US) and includes: all of the shareware disks, including each new one as it comes out; a subscription to GEOWORLD; full access to Mystic Jim's 20M BBS, with its growing program library, games, contests, information, and more; and special discounts on software and hardware. All products carry a money-back guarantee and none of the software is copy-protected. The BBS provides customer service.

Programmers are invited to submit their programs for inclusion in the shareware library. Mystic Jim makes lump-sum payments for programs that are not in the public domain. Full credit is given for those that are in the public domain.

Mystic Jim, 2388 Grape, Denver, CO, USA 80207. Phone (303) 321-3223 (voice), (303) 321-8954 (BBS), (705) 533-2126 (Canadian BBS).

Update on RomJet Custom Cartridge: In our last issue, we carried an item on the RomJet Custom Cartridges which stated that they were available in sizes ranging from 32K to 256K. In fact, the upper bound of this range is a voluminous 512K. RomJet will install on its cartridges any non-copy-protected programs which you legally own and which permit the creation of back-up copies. For more information, contact: RomJet, 210-2450 Sheppard Ave. E., Willowdale, ON, Canada, M2J 4Z9. Phone (416) 274-7378 or 626-5959.

1988 Commodore Computerfest: The third annual Chicagoland Commodore Computerfest will be held August 28 at the Exposition Center at the Kane County Fairgrounds, St. Charles, IL. The show, presented by the Fox Valley 64 User Group, will feature national speakers, vendors, and products for the 64, 128, and the Amiga. It is the largest Commodore computer club show in the midwest. Admission fee is $5.00 for the day and includes access to all the speaker and technical sessions. For more information, write to: Computerfest, P.O. Box 28, North Aurora, IL, 60542.

Superboot for C128: Superboot is software that lets you create your own auto-boot disks that will run your program in either C128 or C64 mode when the system is booted. Available from: JT Program Software, 100 North Beretania St., Suite 210, Honolulu, HI, USA, 96817.

Computer Save is an independent monthly publication designed to provide assistance to buyers and sellers of quality orphan equipment. They also advertise for both manufacturers and retailers of the newest hardware. Their aim is to inform and entertain by way of constantly updated press releases and feature articles by writers well versed in their particular fields, whether the very newest or the orphans. Computer Save is even now planning to expand their aid by way of new and exciting additions to their format. Watch for future issues. Contact: Elizabeth Hartwell, 278-3017 St. Clair Ave., Burlington, ON, Canada, L7N 3P5. Phone (416) 529-0580.

Satellite Tracking program for the C64/C128: SATCOMM-64 allows Amateur radio operators or others using communications satellites to track up to 15 different satellites, and provides key data at user-selected intervals of one minute or more. The user can select screen-only searches, or generate printed reports so that the computer is available for communications use during actual satellite passes. The printed reports include: relative azimuth and elevation, actual altitude, longitude and latitude, local time, UTC day, geographic areas that are within the satellite's communication range, doppler shift, minimum and maximum communication distance, operating frequencies, orbit number, and phase.

SATCOMM-64 overcomes traditional satellite tracking program shortcomings with features like annual rollover, standard-to-daylight time change-over, and single setup multi-day/multi-satellite reports. The program comes with data for several amateur radio, visible, and weather/research satellites; whenever desired, the user can replace these with new satellite choices.

SATCOMM-64 is compatible with the C64/128, 1541 disk drive and 1525 printer, and is available for $15.95 (MO residents add tax) plus $3.00 p&h from: Strategic Marketing Resources, Inc., P.O. Box 2183, Ellisville, MO 63011. Phone (314) 256-7814.

Micro Detective professional debugger for the C64 and C128: Micro Detective is a resident debugging facility that provides interactive trace modes, advanced program error detection and reporting, and programmers' utility commands. The trace can be turned on or off at will while a BASIC program executes, and operates on a separate screen so that the display of the program being traced is not interfered with. The C128 version displays trace information in a separate window anywhere on the 40 or 80 column screen. Conditional tracing allows you to trace only certain program lines, variables, statements, or when certain conditions are met.

Micro-Detective's error detection gives specific, clear error messages instead of the standard '?syntax error' or other system message. More meaningful messages, like "Expected a comma", or "Variable must start with a letter" help the programmer spot the problems much more quickly. Micro detective displays the section of code that caused the error, and handles all kinds of problems, including numeric overflow and disk errors.

Micro Detective also provides a complete set of programmers' aids: bidirectional program scrolling through program listings; AUTO, DELETE, DIR, DISK, RENUM, etc.; variable cross reference list; disk commands; program merging; move ranges of program lines; SLIST, which lists a program with spaces in intelligent places to make it more readable; plus many other commands and features (a total of over 30 new commands are added).

Micro Detective for the C64, with everything mentioned above, is $49.95 (US). In the C128 version, the debugger comes without the error detection feature, for the same price; the C128 error detection program is available separately. From: American Made Software, P.O. Box 323, Loomis, CA 95650.

The Anatomy of the 4040 Disk Drive, written and published by Hilaire Gagne, is filled with memory maps, ROM routine explanations, disassembled source code, technical details and other hard facts about the 4040. Cost is $39.95 (CDN) for Canadian residents, plus $3 shipping and handling; In the U.S., $31.95 (US) plus $9 shipping and handling. Order from: Hilaire Gagne, 4501 Carl St., P.O. Box 278, Hanmer, Ont., P0M 1Y0.

Free Spirit releases C64 version of Super 81 Utilities: Free Spirit Software has released a version of Super 81 utilities for the C64. Now you can copy whole disks or files from 1541 or 1571 disk drives to the 1581. It also backs up disks or files with one or two 1541s, one or two 1581s, or any combination. Also included is a full-featured sector editor, partitioning utilities, scratch/unscratch, lock/unlock, and other file utilities.

Super 81 Utilities is supplied on both 5 1/4" and 3 1/2" diskettes and will boot from device 8 or 9. The package costs $39.95 (US) - shipping/handling are free. For more information, contact: Joe Hubbard, Free Spirit Software, Inc., 905 W. Hillgrove, Suite 8, La Grange, IL 60525. Phone 1-800-552-6777.

CP/M Starter Set from Public Domain Solutions: The newest product from Public Domain Solutions for the C128 is the PDS CP/M Starter Set. This set consists of four disks full of CP/M utilities, plus printed documentation which explains: The history of CP/M; Booting up; Transient commands; Resident commands; Creating and dissolving library (LBR) files; How to run software on the CP/M operating system. The set is $29.95 (US). Order toll-free 1-800-634-5546 or write to: Public Domain Solutions, CP/M Dept., P.O. Box 832, Tallevast, FL 34270.

place
stamp
here

Transactor
85-10 West Wilmot Street
Richmond Hill, Ontario,
Canada

Tranzblooperz

"The Projector Part II" had an error in program Listing 1, on page 21. Line 2220 read:

```
2220  an = th * <clr> / 180
```

it should have read:

```
2220  an = th * ¶ / 180
```

"¶" represents the "pi" symbol, entered with the shift/up-arrow key sequence. In upper/lowercase mode, the symbol will appear as a small checkerboard-like character.

Our thanks to Don Lokken of Minneapolis, MN for spotting and reporting this error.

To order Transactor products or magazine/disk subscriptions in the U.S., use our new

TOLL-FREE ORDER LINE

1-800-248-2719 Extension 911

(For placing orders only)

Reader Service Card

Fill out this card (please print) or attach your peel-off mailing label, and circle the ads below that you'd like more information about.

Name _____

Company _____

Address _____

_____

Postal/Zip Code _____

Advertiser	Page
Brown Boxes Inc.	29
Dave Taylor Enterprises	1
Free Spirit Software Inc.	29
NorthWest Music Centre Inc	IFC
Schnedler Systems	1

Transactor Classifieds

...are a great way o reach thousands of Commodore microcomputer enthusiasts inexpensively - just $2 per word! Send your ad copy with cheque or money order to: Transactor Classifieds, 85-10 West Wilmot Street, Richmond Hill, Ontario, Canada L4B 1K7.

FOR SALE: SFD 1001 $129; 8250 $185; 8050 $145; MSD $310; 8023 $99; 6400 $199; PET 64 $99 Plus shipping.

BETHUREM ENTERPRISES (303) 693-4038 (Prices in US$).

Transactor Bimonthly Special Offer

Our "5-for-3" disk deal was such a success the last time that we're offering it again. Order any five regular Transactor magazine issue disks from the card at right (priced at $8.95 US/$9.95 Cdn) and pay for just three of them - that's five disks full of Utilities and Transactor programs, for just $26.85 US or $29.85 Cdn! Note that this offer does not apply to special disks like "Bits Book" or TransBasic disks.

Transactor

The Magazine for Commodore Programmers

Please print your name and address below (or transfer your mailing label)
Check here for change of address ☐

Name: ___ Sub#4880 EXP. Vol. 9 Iss. 5 _____
Address: ___ Boyd Ray
18605 66 Avenue _____
City: ___ Edmonton, AB
Canada T5T 2M3 _ *Province/State:* _____

Zip/Postal: _____ *Phone:* _____ *CompuServe ID:* _____

IMPORTANT NOTE: Prepayment is required on ALL orders

Subscribers only: please enter subscription number from mailing label ☐

Subscription Order Section

	Canada ($Cdn.)			U.S.A. ($US)			Foreign ($US)			Air Mail ($US, overseas only)		
	Mag	Disk*	Both*	Mag	Disk	Both	Mag	Disk	Both	Mag	Disk	Both
1 year	☐ 19.00	☐ 55.00	☐ 74.00	☐ 15.00	☐ 45.00	☐ 60.00	☐ 21.00	☐ 55.00	☐ 76.00	☐ 40.00	☐ 69.00	☐ 109.00
2 years	☐ 35.00	☐ 101.00	☐ 136.00	☐ 27.50	☐ 82.50	☐ 110.00	☐ 38.50	☐ 101.50	☐ 139.50	☐ 73.00	☐ 126.50	☐ 199.50
3 years	☐ 47.50	☐ 137.50	☐ 185.00	☐ 37.50	☐ 112.50	☐ 150.00	☐ 52.50	☐ 137.50	☐ 190.00	☐ 100.00	☐ 172.50	☐ 272.50

*** Ontario residents please add 8% of disk subscription price for provincial sales tax. No PST on magazine subscriptions.**

☐ **Please send Microfiche instead of magazines** ☐ **Send me info on Transactor for the Amiga**

Combo Subscription Bonus!

If you are subscribing to BOTH the magazine and the disk, check one of the following FREE gifts:

☐ **TransBASIC 2 Disk and Manual** ☐ **Potpourri Disk** ☐ **Extended Subscription (extra issue)**

Product Order Section

Ontario residents please add 8% PST on all items

$Cdn.	$U.S.		$Cdn.	$U.S.	
☐ 17.95	14.95	Transactor Book of Bits and Pieces #1	☐ 69.95	59.95	G-LINK C64 to IEEE Interface
☐ 9.95	8.59	Bits and Pieces Book Disk	☐ 19.95	17.95	Potpourri Disk
☐ 24.95	19.95	Bits and Pieces Book AND Disk Combo	☐ 129.95	99.95	The Micro Sleuth
☐ 19.95	17.95	The TransBASIC 2 Disk	☐ 89.00	69.00	16K QBB - (Please add $3 US,
☐ 19.95	17.95	Jugg'ler 128 CP/M Formatting Utility	☐ 129.00	99.00	32K QBB - $4 Cdn. postage and handling
☐ 59.95	49.95	Transactor Microfiche Set (Vols 4 through 8)	☐ 169.00	129.00	64K QBB - for any Quick Brown Box)

The Transactor Disk, $8.95 US, $9.95 Cdn each (1541/4040/2031/MSD format)

Numbers shown are Disk numbers; the Volume and Issue of the corresponding magazine are shown in brackets

☐ 1 (V4, #1-6)	☐ 4 (V5, #5)	☐ 7 (V6, #2)	☐ 10 (V6, #5)	☐ 13 (V7, #2)	☐ 16 (V7, #5)	☐ 19 (V8, #2)	☐ 22 (V8, #5)
☐ 2 (V5, #1-3)	☐ 5 (V5, #6)	☐ 8 (V6, #3)	☐ 11 (V6, #6)	☐ 14 (V7, #3)	☐ 17 (V7, #6)	☐ 20 (V8, #3)	☐ 23 (V8, #6)
☐ 3 (V5, #4)	☐ 6 (V6, #1)	☐ 9 (V6, #4)	☐ 12 (V7, #1)	☐ 15 (V7, #4)	☐ 18 (V8, #1)	☐ 21 (V8, #4)	☐ 24 (V9, #1)

Add flat $2.00 shipping & handling for any number of items ordered from above box.

Transactor Back Issues and Microfiche: $4.50 U.S., $4.50 Cdn. (MF = Microfiche)

☐ V4, #1	☐ MF	☐ V4, #6	MF only	☐ V5, #5	☐ MF	☐ V6, #4	☐ MF	☐ V7, #3	☐ MF	☐ V8, #2	☐ MF
☐ V4, #2	☐ MF	☐ V5, #1	☐ MF	☐ V5, #6	☐ MF	☐ V6, #5	☐ MF	☐ V7, #4	☐ MF	☐ V8, #3	☐ MF
☐ V4, #3	MF only	☐ V5, #2	MF only	☐ V6, #1	☐ MF	☐ V6, #6	☐ MF	☐ V7, #5	☐ MF	☐ V8, #4	☐ MF
☐ V4, #4	MF only	☐ V5, #3	MF only	☐ V6, #2	☐ MF	☐ V7, #1	☐ MF	☐ V7, #6	☐ MF	☐ V8, #5	☐ MF
☐ V4, #5	MF only	☐ V5, #4	MF only	☐ V6, #3	☐ MF	☐ V7, #2	☐ MF	☐ V8, #1	☐ MF	☐ V8, #6	☐ MF

Ontario residents please add 8% provincial sales tax for any items ordered from this box.

☐ Visa ☐ MasterCard ☐ Cheque/MO enclosed Cheque #_____ Dated ____/____/____ Amount _____

Acct. # _____ Expires ____/____ Signature _____

Please allow 6-8 weeks for delivery in the U.S., 4-6 weeks in Canada.

FOLD ALONG SCORES WITH CORRECT REPLY CARD ON OUTSIDE. USE TAPE, PLEASE - NO STAPLES OR GLUE.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 390 BUFFALO, NY

POSTAGE WILL BE PAID BY ADDRESSEE

The Transactor
P.O. Box 338 Station 'C'
Buffalo, NY, 14209-9990

Business Reply Mail
No Postage Stamp Necessary
if mailed in Canada

Postage will be paid by:

Canada Post Postes Canada
46487

The Transactor
501 Alden Road
P.O. Box 3250
Markham Industrial Park
Markham, Ontario
L3R 9Z9

The Potpourri Disk

Help!

This HELPful utility gives you instant menu-driven access to text files at the touch of a key – while any program is running!

Loan Helper

How much is that loan really going to cost you? Which interest rate can you afford? With Loan Helper, the answers are as close as your friendly 64!

Keyboard

Learning how to play the piano? This handy educational program makes it easy and fun to learn the notes on the keyboard.

Filedump

Examine your disk files FAST with this machine language utility. Handles six formats, including hex, decimal, CBM and true ASCII, WordPro and SpeedScript.

Anagrams

Anagrams lets you unscramble words for crossword puzzles and the like. The program uses a recursive ML subroutine for maximum speed and efficiency.

Life

A FAST machine language version of mathematician John Horton Conway's classic simulation. Set up your own 'colonies' and watch them grow!

War Balloons

Shoot down those evil Nazi War Balloons with your handy Acme Cannon! Don't let them get away!

Von Googol

At last! The mad philosopher, Helga von Googol, brings her own brand of wisdom to the small screen! If this is 'AI', then it just ain't natural!

News

Save the money you spend on those supermarket tabloids – this program will generate equally convincing headline copy – for free!

Wrd

The ultimate in easy-to-use data base programs. WRD lets you quickly and simply create, examine and edit just about any data. Comes with sample file.

Quiz

Trivia fanatics and students alike will have fun with this program, which gives you multiple choice tests on material you have entered with the WRD program.

AHA! Lander

AHA!'s great lunar lander program. Use either joystick or keyboard to compete against yourself or up to 8 other players. Watch out for space mines!

Bag the Elves

A cute little arcade-style game; capture the elves in the bag as quickly as you can – but don't get the good elf!

Blackjack

The most flexible blackjack simulation you'll find anywhere. Set up your favourite rule variations for doubling, surrendering and splitting the deck.

File Compare

Which of those two files you just created is the most recent version? With this great utility you'll never be left wondering.

Ghoul Dogs

Arcade maniacs look out! You'll need all your dexterity to handle this wicked joystick-buster! These mad dog-monsters from space are not for novices!

Octagons

Just the thing for you Mensa types. Octagons is a challenging puzzle of the mind. Four levels of play, and a tough 'memory' variation for real experts!

Backstreets

A nifty arcade game, 100% machine language, that helps you learn the typewriter keyboard while you play! Unlike any typing program you've seen!

All the above programs, just $17.95 US, $19.95 Canadian. No, not EACH of the above programs, ALL of the above programs, on a single disk, accessed independently or from a menu, with built-in menu-driven help and fast-loader.

The ENTIRE POTPOURRI COLLECTION
JUST $17.95 US!!

See Order Card at Center

THE TIME SAVER

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only $8.95 US, $9.95 Cdn. Per Issue
6 Disk Subscription (one year)
Just $45.00 US, $55.00 Cdn.
(see order form at center fold)

Now Amiga Owners Can Save Time Too!

Transactor Amiga Disk #1, $12.95 US, $14.95 Cdn.

All the Amiga programs from the magazine, with complete documentation on disk, plus our pick of the public domain!